Kübler



Handbuch

Sendix 5858/5878 absolut singleturn
Sendix 5868/5888 absolut multiturn

Für Bestellschlüssel 8.58X8.XXCX.C2XX
ab Firmware-Version 2.0

Für Bestellschlüssel 8.58X8.XXCX.C1XX
ab Firmware-Version 1.37

wir geben Impulse

Kübler

Änderungsvorbehalt
Änderungen der in dem vorliegenden Dokument enthaltenen technischen Informationen, die aus dem stetigen Bestreben zur Verbesserung unserer Produkte resultieren, behalten wir uns jederzeit vor.

Verzicht auf Garantie
Die Fritz Kübler GmbH übernimmt in Bezug auf das gesamte Handbuch keine Garantie, weder stillschweigend noch ausdrücklich und haftet weder für direkte noch indirekte Schäden. Angegebene Produkteigenschaften und technische Daten stellen keine Garantieerklärung dar.

Dokumenteninformation
Ausgabestand 07/2013
Originalhandbuch, Deutsch ist die Originalfassung.

Kübler Group
Fritz Kübler GmbH
Schubertstraße 47
78054 Villingen-Schwenningen
Deutschland
Tel.: +49 7720 3903-0
Fax: +49 7720 21564
info@kuebler.com
www.kuebler.com

Inhaltsverzeichnis

Version der Firmware und der GSDML-Datei

Die Versionen der Firmware und der GSDML-Datei zum Release-Zeitpunkt dieser Dokumentation sind:
Firmware-Version V2.00 GSDML-V2.2-KUEBLER-0198-Sendix58xxPNIO-20130116-131800.xml

Technische Details und Drehgebereigenschaften

Mechanische Kennwerte

Schockfestigkeit nach EN 60068-2-27	2500 m/s², 6ms für Singleturn 2000 m/s², 6ms für Multiturn
Vibrationsfestigkeit nach EN 60068-2-6	100m/s² , 10.....2000 Hz

Arbeitstemperaturbereich
-40…+85°C

Versorgungsspannung und Stromverbrauch
10…30 VDC

200 mA bei 10 VDC
80 mA bei 24 VDC
60 mA bei 30 VDC

Hardware-Eigenschaften
PROFINET IO ASIC: ERTEC 200
Auto-Negotiation
Auto-Polarity
Auto-Crossover
Funktionsanzeige und Diagnostik mittels LEDs

Unterstützte Standards und Protokolle
RT_CLASS_1
RT_CLASS_2
RT_CLASS_3 (IRT)
DCP
RTA
LLDP
SNMP
MIB-II und LLDP-MIB
PTCP
MRP

Implementiertes Drehgeber-Profil
Version 4.1

Identification- und Maintenance-Funktionalität
Version 1.2
Unterstützte I&M Blöcke 0, 1, 2, 3, 4

Konformität gewährleistet entsprechend
EN 61000-4-2 :2001
EN 61000-4-3 :2006
EN 61000-4-4 :2005
EN 61000-4-5 :2007
EN 61000-4-6 :2008
EN 61000-4-7 :2004

EN 61000-6-4 :2007
EN 61000-6-2 :2006

Installation

Die Inbetriebnahme des Drehgebers als Teil einer Anlage besteht aus fünf Schritten:

1. Installation der Verkabelung. Hierzu Referenz [1] beachten.
2. Installation der Spannungsversorgung.
3. Installation des Drehgebers und der Steuerung. Hierzu Referenzen [2] und [3] beachten.
4. Projektierung mit STEP 7
5. Start der Applikation

Installation der Verkabelung
Der Drehgeber hat drei Anschlüsse, von denen zwei die beiden Ethernet-Ports sind. In dieser Dokumentation werden sie als Port 1 und Port 2 referenziert. In der unteren Abbildung 1 sind es die beiden Pfeile „PORT 2“ und „PORT 1“ des Aufklebers, die die Positionierung angeben. Beim mittleren Anschluss handelt es sich um den Spannungsversorgungsanschluss, der im nächsten Kapitel beschrieben wird.



Abbildung 1

Die beiden Ethernet-Anschlüsse sind D-kodierte M12-Buchsen. Die Zuordnung der Signale zu den Pins ist in Abbildung 2 und der nachfolgenden Tabelle dargestellt.

Signalzuordnung einer D-kodierten M12-Buchse

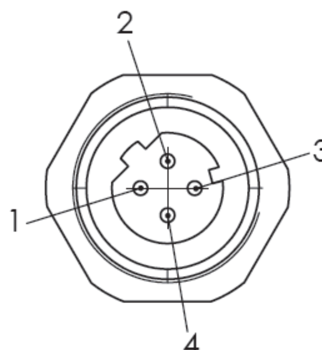


Abbildung 2: D-kodierte M12-Buchse des Drehgebers

Signalname einer M12 D-kodierten Buchse	Funktion	Litzenfarbe	Pin-Nummer
TD+	Transmit data +	Gelb	1
TD-	Transmit data -	Orange	3
RD+	Receive data +	Weiß	2
RD-	Receive data -	Blau	4

Signalzuordnung eines RJ45 zu M12 – Kabels
M12 zu RJ45 direkt

Signal	M12-Pin-Nummer	RJ45-Pin-Nummer
TD+	1	1
TD-	3	2
RD+	2	3
RD-	4	6

M12 zu RJ45 crossover

Signal	M12-Pin-Nummer	RJ45-Pin-Nummer
TD+	1	3
TD-	3	6
RD+	2	1
RD-	4	2

Empfohlenes Kabel für PROFINET-Netzwerk-Verkabelung:
Siemens Industrial Ethernet FC TP flexible Cable,
GP 2x2 (PROFINET Type B), Twisted Pair Installation
Bestellnummer: 6XV1870-2B

Empfohlener RJ45-Stecker:
Siemens IE FC RJ45 Bestellnummer: 6GK1901-1BB10-2AA0

ACHTUNG!



Da PROFINET auf Fast Ethernet – Technologie basiert, ist darauf zu achten, dass die Segmentlänge auf keinen Fall 100 m überschreitet. Für Längen größer als 100m müssen Switches dazwischen geschaltet werden. Auch hier ist darauf zu achten, dass keine Hubs zum Einsatz kommen ! Die eingesetzten Switches müssen PROFINET-zertifiziert sein, da sie Anforderungen an PROFINET-Protokolle erfüllen müssen. Wird der Drehgeber für den MRP-Betrieb konfiguriert (Media-Redundancy-Protocol), dann müssen die dazwischengeschalteten Switches sogar gemanaged sein und im Falle von IRT-Mode auch IRT-fähig sein ! Beispiel für einen Switch, der sowohl managed als auch IRT-fähig ist, ist der SCALANCE 200 IRT.

Installation der Spannungsversorgung

Abbildung 3 und die nachfolgende Tabelle zeigen die Signalzuordnung zu den Pins eines A- kodierten Netzanschluss-Steckers am Drehgeber.

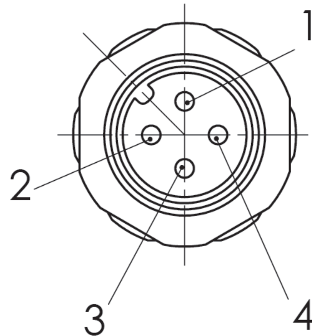


Abbildung 3: A-kodierter M12-Stecker des Drehgebers

Signal am M12- Stecker A-kodiert	**Funktion**	**Pin-Nummer**
PWR	10 – 30 V DC	1
		2
GND	0V	3
		4

In der Referenz [1], PN-Cabling-Guide_2252_V200_May07.pdf, sind mehr Details zu PROFINET-Installationen zu finden. Die dort beschriebenen Anforderungen sind unbedingt einzuhalten.

Diese Spezifikation kann von der Web-Page http://www.profibus.com/downloads/ heruntergeladen werden. Es handelt sich um einen Bereich, der auch Nicht-Mitgliedern der PNO zugänglich ist.

Diagnostik-LEDs

Es gibt am Drehgeber vier Diagnostik – LEDs, deren Funktionalität in der Folge beschrieben wird.



Abbildung 4

LED-Bezeichnung	Farbe	Funktionsbeschreibung
LINK 1	Gelb und Grün	LINK 1 ist eine Bicolour-LED für Port 1, die sowohl Link- (Grün) als auch Datentransfer- Aktivität (Gelb) anzeigt
LINK 2	Gelb und Grün	LINK 2 ist eine Bicolour-LED für Port 2, die sowohl Link- (Grün) als auch Datentransfer- Aktivität (Gelb) anzeigt

Die folgende Tabelle beschreibt die Betriebsmodi eines PROFINET-Drehgebers, die aus dem Zustand der ERROR- und PWR-LED abgeleitet werden.

ERROR LED (rot)	PWR LED (grün)	Bedeutung	Mögliche Ursache
Aus	An	Normaler Betriebsmodus. Datenaustausch ordnungsgemäß.	
Blinkt	An	Datenaustausch prinzipiell möglich, allerdings konnte der Drehgeber nicht in den „Operational“-Modus umschalten. Der Drehgeber zeigt einen Fehler in G1_XIST2. In Abhängigkeit des Fehlertyps kann er auch als Teil des Parameters 65001 aus dem Drehgeber ausgelesen werden.	Siehe Fehler- Blink-Codes im nächsten Kapitel
An	An	Datenaustausch prinzipiell möglich, allerdings findet kein Datenaustausch über den Bus statt	Master nicht vorhanden oder Drehgeber nicht am Bus
Aus	Aus	Drehgeber nicht bestromt	

Fehler-Blink-Codes

Blink-Code	Ursache
Ein Mal alle 2 Sekunden (0.5Hz) Error LED An für 1 sec., Aus für 1 sec., danach Wiederholung	- Drehgeber wurde noch nicht vom Master konfiguriert. Die Benutzer-Parameterdaten im Sinne des Indexes 0xBF00 wurden noch nicht vom Drehgeber empfangen. - Falsche Konfiguration - Falsche Stationsadresse zugewiesen (wenn auch innerhalb des zulässigen Bereichs) - Die aktuelle Konfiguration des Slaves weicht von der nominalen Konfiguration ab
5 Mal in der Sekunde Error LED An für 0.1 sec., Aus für 0.1 sec., danach Wiederholung	Bus-Kommunikation in Ordnung, allerdings hat der Drehgeber die Verbindung zum Positionsdaten-Sensor verloren
Ein Mal pro Sekunde (1Hz) Error LED An für 0.5 sec., Aus für 0.5 sec., danach Wiederholung	Speicherfehler

Konfiguration eines Beispielprojektes mit STEP 7

Wichtig!



Für die Projektierung muss unbedingt die Version V5.5 von STEP7 benutzt werden. Anderenfalls kommt es zu Fehlern bei der Installation der Drehgeber-GSDML-Datei und allen anderen GSDML-Dateien, die dasselbe XML-Schema benutzen. Des Weiteren kommt es zu Fehlern bei der Konfiguration und Parametrierung von MRP.

Ein STEP7-Beispielprojekt namens „Kuebler_Sample_Project“, das in der Folge verwendet wird, kann vom Kübler-Web-Server heruntergeladen werden. Es befindet sich dort außerdem die GSDML-Datei des MRP-fähigen PROFINET-Drehgebers, die vor der Inbetriebnahme des Drehgebers unter STEP7 zu installieren ist.

Die folgenden zwei Abbildungen repräsentieren das Projekt selbst und die Hardware- Konfiguration mit zwei Drehgebern, die alle denselben DAP verwenden, wenn auch mit unterschiedlichen Modulen.

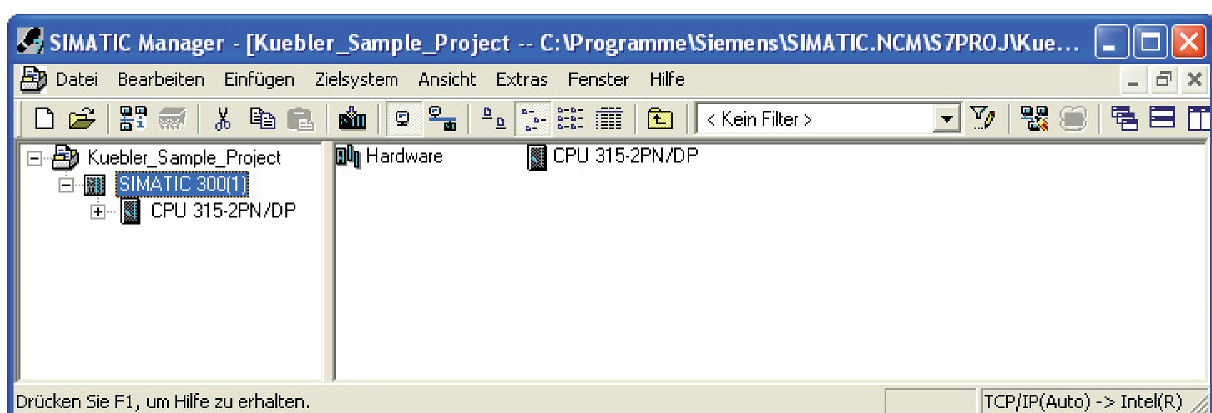


Abbildung 5

Der erste Drehgeber dg1 verwendet das Modul mit dem Standard Telegramm 81 , was in Abbildung 6 sichtbar ist. Seine Adresse zum Lesen des Standard-Telegramms 81 ist 0. Die Schreib-Adresse für das Standard-Telegramm 81, mit dessen Hilfe unter anderem das Preset ausgelöst wird, ist ebenfalls 0.

Der zweite Drehgeber dg2 verwendet das Modul, das sowohl das Standard Telegramm 81 als auch Speed beinhaltet. Die beiden Adressen für das Lesen und Schreiben des Standard- Telegramms 81 sind 12 und 4. Dies zeigt die Abbildung 7.

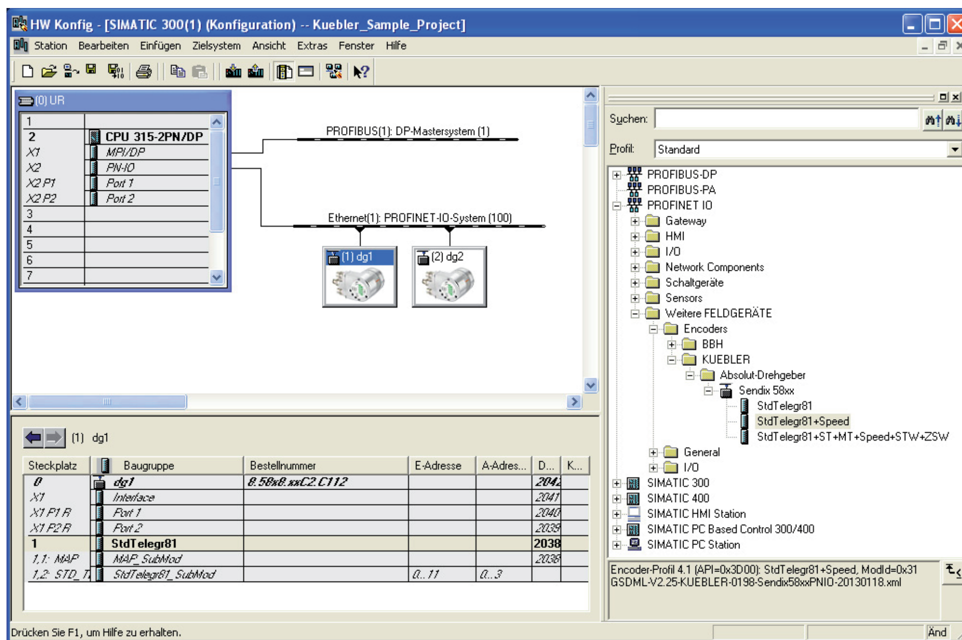


Abbildung 6

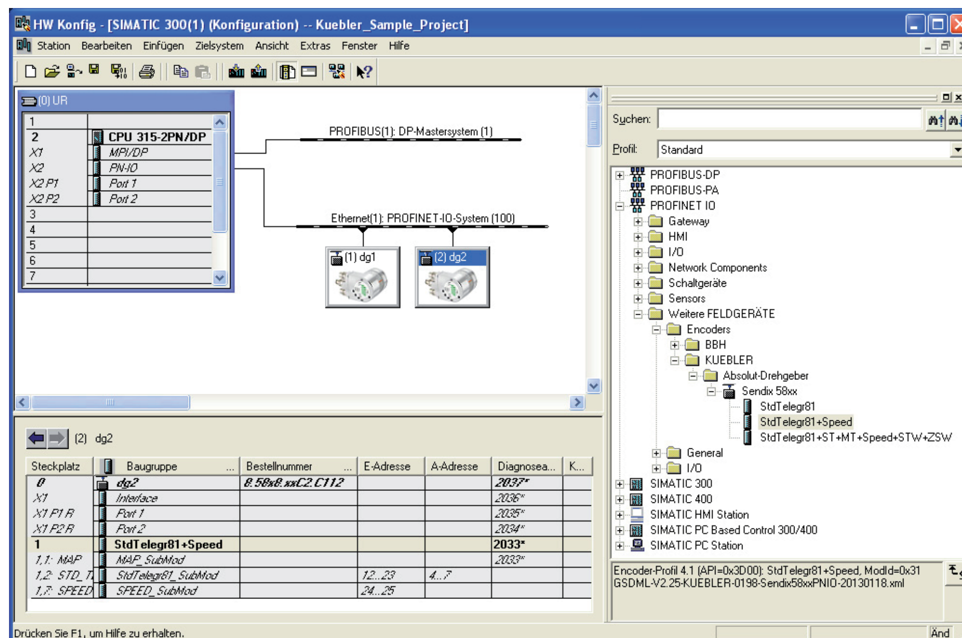


Abbildung 7

Einstellen der Drehgeber-Benutzerparameter

Der Dialog zur Einstellung der Drehgeber-Benutzerparameter gemäß Drehgeber-Profil [2], wird in der obigen Abbildung der Hardwarekonfiguration durch einen Doppelklick auf die Zeile „MAP_SubMod" gestartet. Der Dialog, der sich dadurch öffnet und die einstellbaren Parameter sind in der folgenden Abbildung dargestellt.

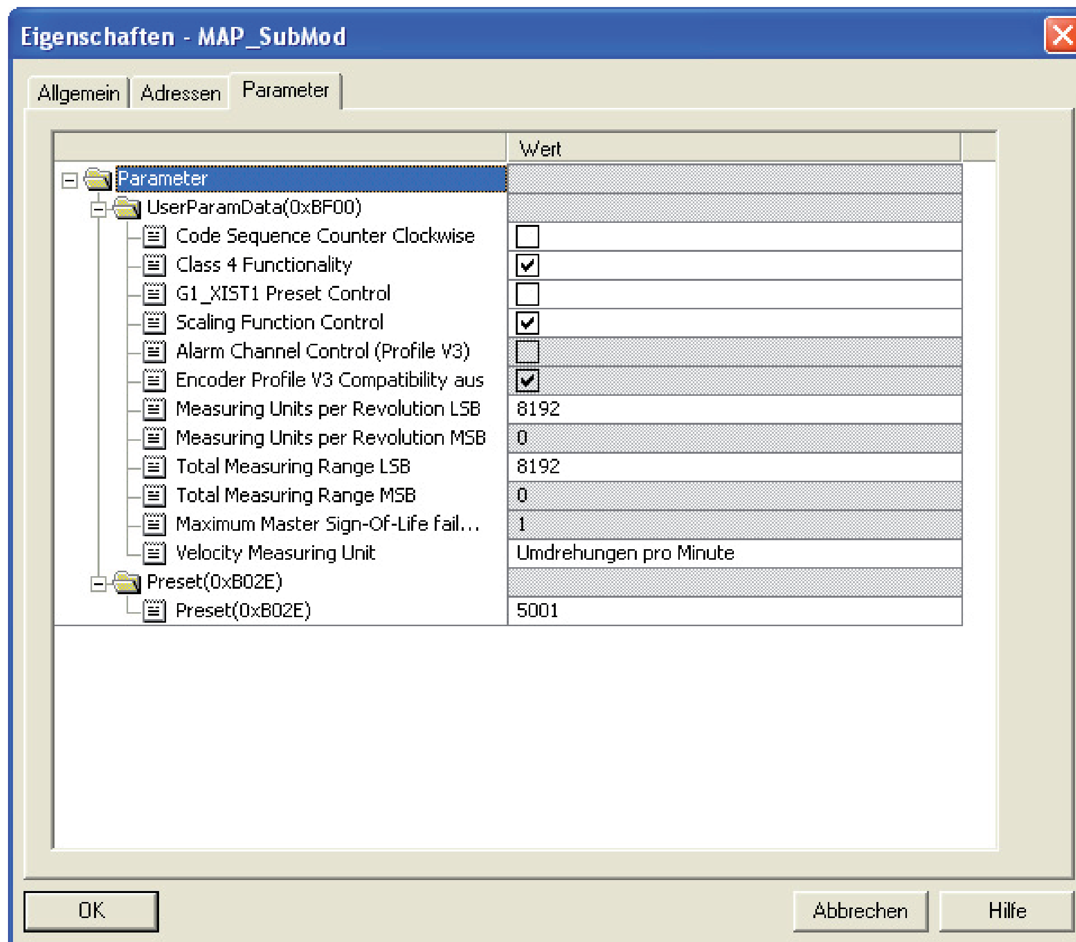


Abbildung 8

Die gewählten Parameter werden von der Steuerung auf den Drehgeber während der Systemanlaufphase übertragen.

Da es sich beim Sendix-Typ des PROFINET-Drehgebers um einen Sensor mit einer 16 Bit Singleturn- und 12 Bit Multiturn – Einheit handelt, sind die zugehörigen vier MSB-Bytes (Most Significant Bytes) von Measuring Units per Revolution (MUR) und Total Measuring Range (TMR) Null, nicht veränderbar und daher grau hinterlegt.

Mit einem Klick auf den Button „OK" werden die gewählten Werte automatisch auf ihren zulässigen Gültigkeitsbereich geprüft. Im Fehlerfall wird der Benutzer dazu aufgefordert, seine Eingaben zu korrigieren, wobei ihm die korrekten Bereiche angezeigt werden.

Die Gültigkeitsbereiche für MUR und TMR werden von folgenden Tatsachen abgeleitet:

- Der MUR-Wert (Measuring Units per Revolution) wird nur dann akzeptiert wenn er das folgende Kriterium erfüllt:
  0 < MUR ≤ g_ST Wobei g_ST
  die physikalische Singleturn-Auflösung darstellt (65536 entsprechend 16Bit).

- Der TMR-Wert (Total Measuring Range) eines Drehgebers **ohne** eine Multiturn-Einheit wird ebenfalls nur dann akzeptiert wenn er das folgende Kriterium erfüllt:
  0 <TMR ≤ g_ST
  Wobei g_ST die physikalische Singleturn-Auflösung darstellt (65536 entsprechend 16Bit).

- Bei einem Drehgeber **mit** einer Multiturn-Einheit muss folgende Bedingung für den TMR- Wert erfüllt sein, damit er akzeptiert wird:
  0 < TMR ≤ MUR*g_MT
  Dabei ist g_MT die physikalische Auflösung der Multiturn-Einheit (4096 entsprechend 12Bit) im Falle des hier beschriebenen Sendix-Typ-Sensors

Achtung!



Die beiden Werte MUR und TMR sollten idealerweise als Vielfache von 2 im Sinne der mathematischen Formel 2^X (2 hoch X) gewählt werden. Nur in diesem Fall ist gewährleistet, dass es keinen „Rest“ dann gibt wenn der Multi-turn-Anteil seinen maximalen Wert erreicht.

Für den Preset-Wert gilt:
- 0 ≤ Preset ≤ TMR-Wert UND
- Preset ≤ 0x7FFFFFFF (MaxINT)

Lesen der Drehgeber-Positionswerte

Die Funktionalität des OB1-Bausteins wurde derart ausprogrammiert, dass sie auf eine sehr einfache Weise das Prinzip des Positionswert-Auslesens verdeutlicht. In der folgenden Abbildung 9 sind es die beiden Netzwerke, die diesen Mechanismus widerspiegeln. Dabei werden von den Adressen 0 und 12 beginnend, jeweils drei Doppelworte, entsprechend der Länge des Standard-Telegramms 81, eingelesen und in DB10 abgespeichert, wo sie einer nachfolgenden Auswertung zur Verfügung stehen.

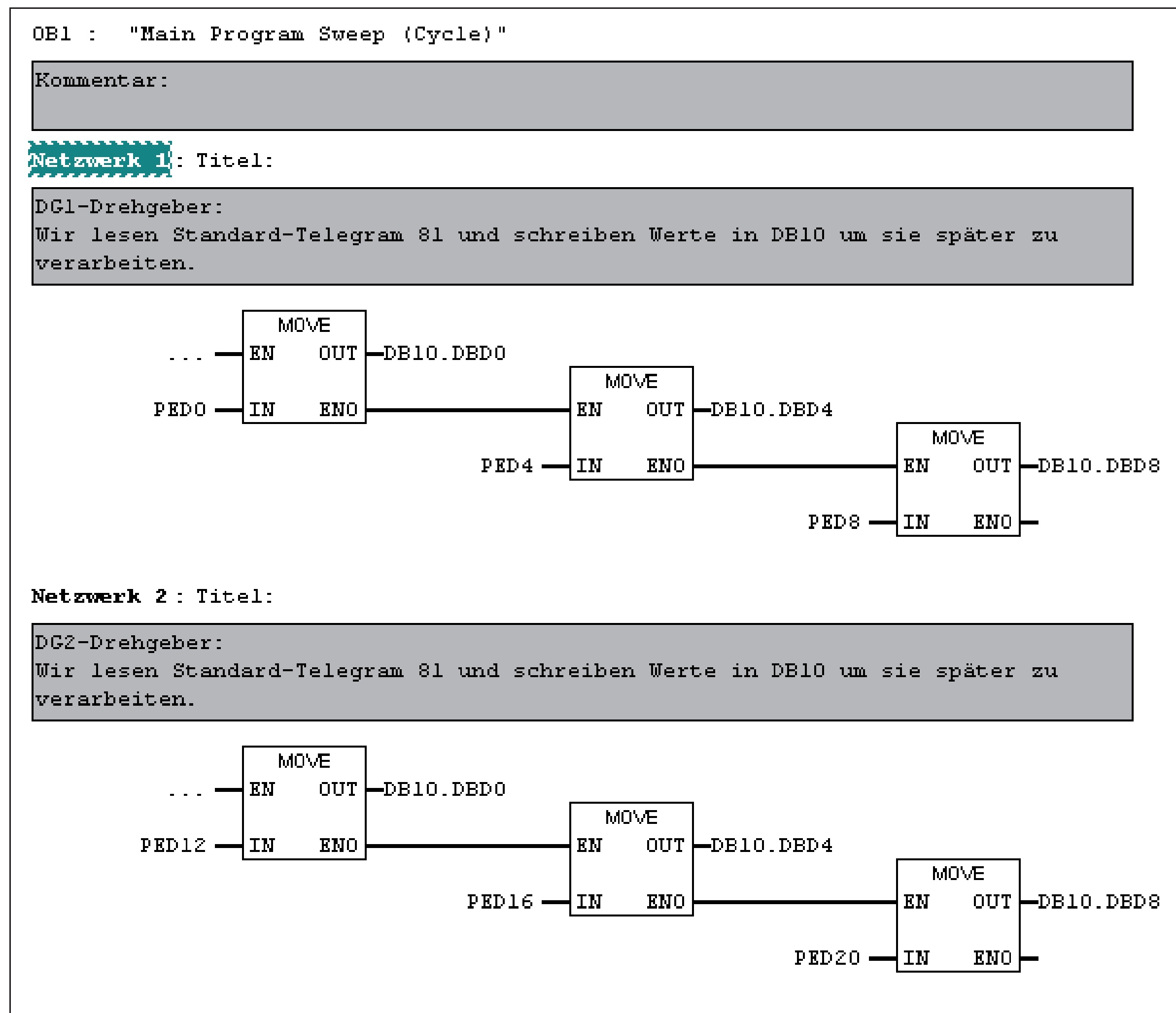


Abbildung 9: OB1-Implementierung mit Lesen der Position

Auslösen des Preset-Vorgangs

Der Preset-Wert ist auch als Parameter 65000 innerhalb des PROFINET-Drehgeber-Profils bekannt. Standardmäßig ist Preset auf den Wert Null voreingestellt, kann jedoch, entsprechend dem Parameter-Dialog aus Abbildung 10, auf einen anderen Wert eingestellt werden. Dabei sind Bedingungen zu beachten, die im vorletzten Kapitel erläutert wurden.

Die Abbildung 10 zeigt das Netzwerk, mit dessen Hilfe auf dem dg1 ein absolutes Preset ausgelöst wird.

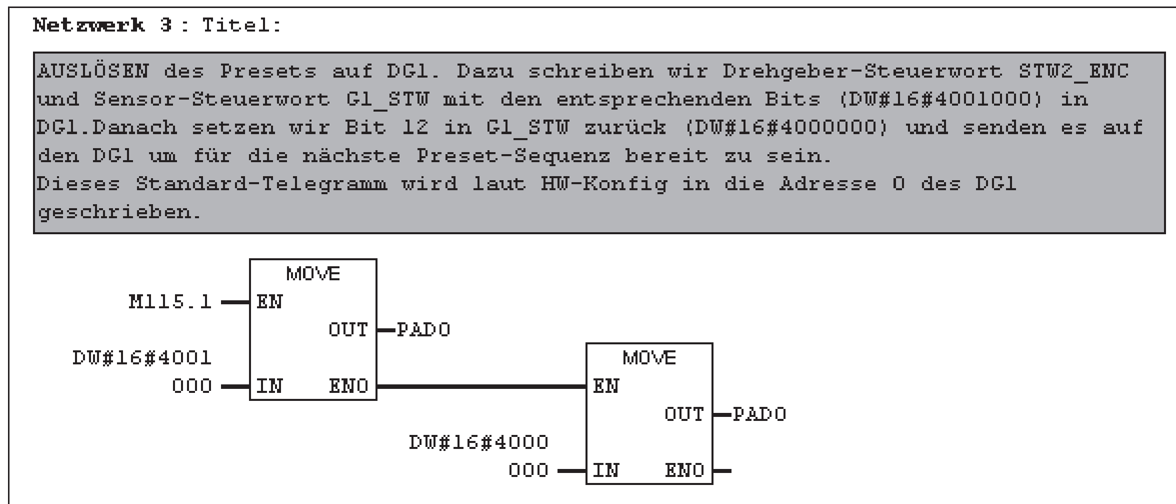


Abbildung 10

Um ein absolutes Preset auszulösen, ist jeweils das Drehgeber-Steuerwort STW2_ENC mit gesetztem PLC-Bit in Position 10 als auch das Sensor-Steuerwort G1_STW mit gesetztem Bit 12 auf den Drehgeber zu senden. Um wiederholt einen Preset-Vorgang auszulösen, muss der Drehgeber zunächst das G1_STW mit zurückgesetztem Bit 12 empfangen.

Für den Drehgeber dg2 gilt die Abbildung 11.

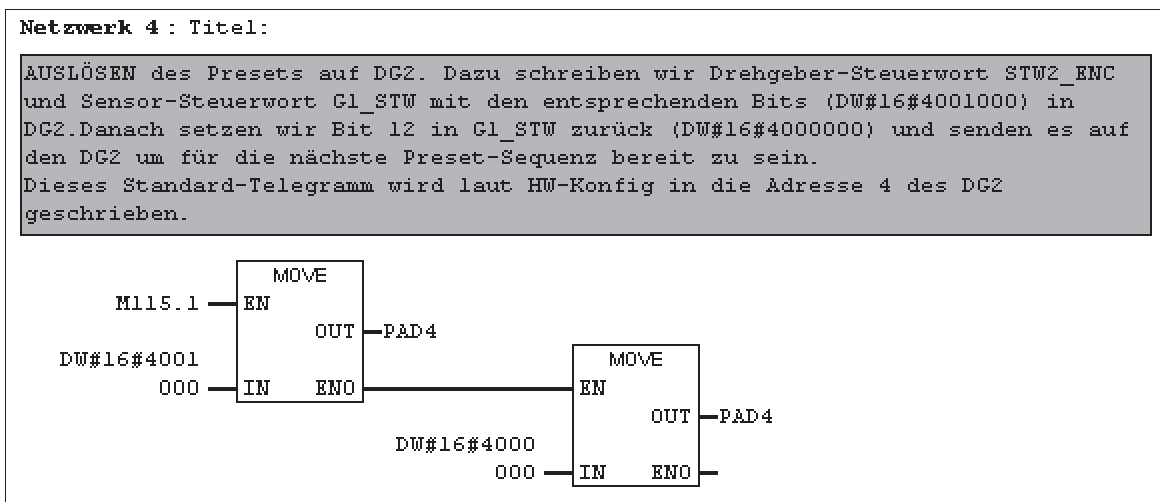


Abbildung 11

Das Auslösen selbst erfolgt in diesem Beispiel über das Bit 1 des Merker-Bytes 115, was in Abbildung 12 in Form einer Variablentabelle dargestellt und über eine Online-Verbindung gesetzt und zurückgesetzt werden kann.

Mehr Details und Informationen zum Drehgeber-Profil selbst können Referenz [2] entnommen werden.

Wie bereits erwähnt, handelt es sich bei dieser Beschreibung um einen rudimentären Ansatz, der das Prinzip verdeutlichen soll. Im konkreten Fall einer Anlagenprogrammierung obliegt es dem Leser selbst entsprechende Änderungen vorzunehmen.

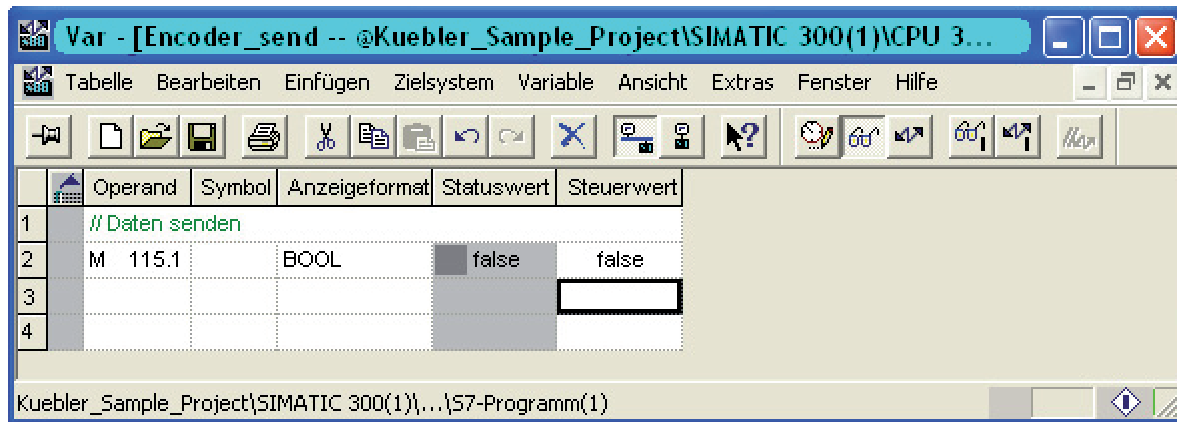


Abbildung 12

Download des Parameters 65000 (Preset)

Das vorhergehende Kapitel behandelt das Auslösen des Preset-Vorgangs auf dem Drehgeber. Der Download dieses Wertes ist Gegenstand dieses Kapitels.

Der Preset-Wert wird zum Download-Zeitpunkt nur dann akzeptiert wenn er die beiden folgenden Kriterien erfüllt:

0 ≤ Preset ≤ TMR (SubIdx 10 Parameter 65001) und Preset ≤ MaxINT32

Gemäß Spezifikation basiert der Preset-Wert auf skalierten Einheiten, weshalb sowohl die Checkbox „Scaling function control“ als auch „Class 4 Functionality“ im Dialog der Abbildung 10 gecheckt sein müssen und der Drehgeber mit diesen Einstellungen in Betrieb gehen muss.

Neben der Möglichkeit, den Preset-Wert über die Steuerung, gemäß Abbildung 8 auf den Drehgeber zu senden, gibt es drei weitere Möglichkeiten, die hier näher beschrieben werden sollen.

Setzen des Preset-Wertes mittels Kuebler-Step7-FB-Bausteins
Auf dem Kübler- Web-Server befindet sich die Bibliothek „Kuebler_Library.zip“, die es erlaubt den Preset-Wert in den Drehgeber an jeder beliebigen Stelle des Steuerprogramms zu schreiben. Gehen Sie bei der Installation dieser Bibliothek gemäß den folgenden Abbildungen 13 bis 15 vor. Im Einzelnen bedeutet dies, dass Kuebler_Library.zip in das Verzeichnis S7libs des Step7 – Installationsverzeichnisses dearchiviert wird.

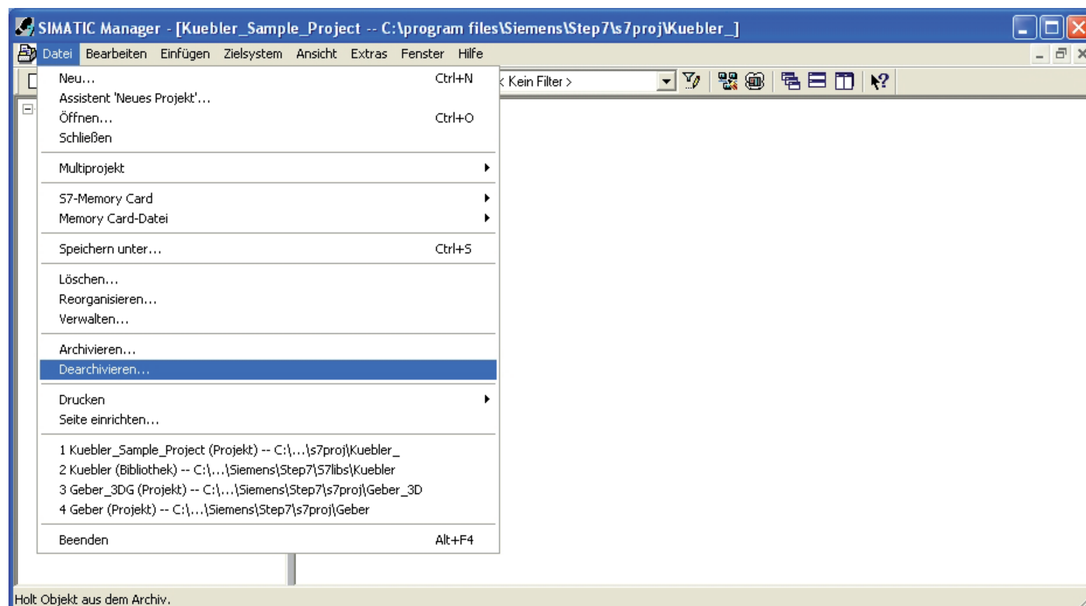


Abbildung 13

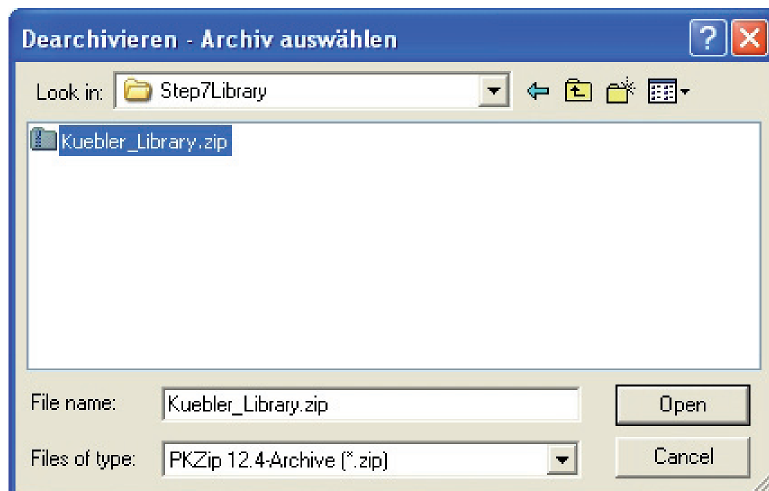


Abbildung 14

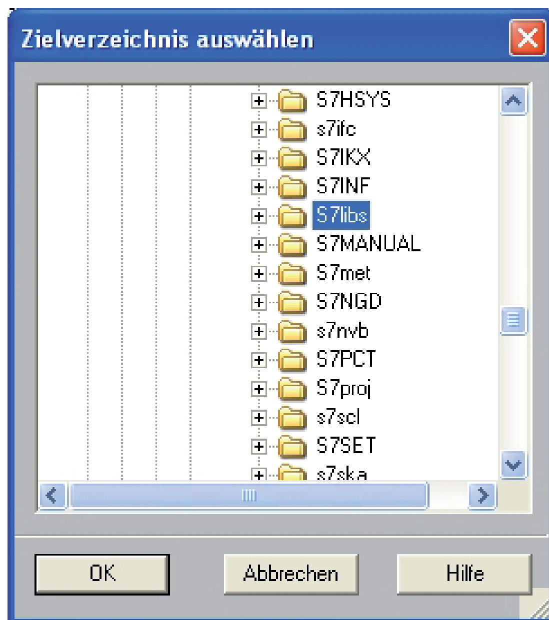


Abbildung 15

Ist die Bibliothek erfolgreich dearchiviert worden, so wird sie beim Öffnen des Baustein- Editors sichtbar und steht zum Einsatz bereit. Dazu siehe folgende Abbildung 16. Insbesondere kann dann der Baustein „FB1 PRESET_ENCODER Kuebler" per Drag&Drop verwendet werden.

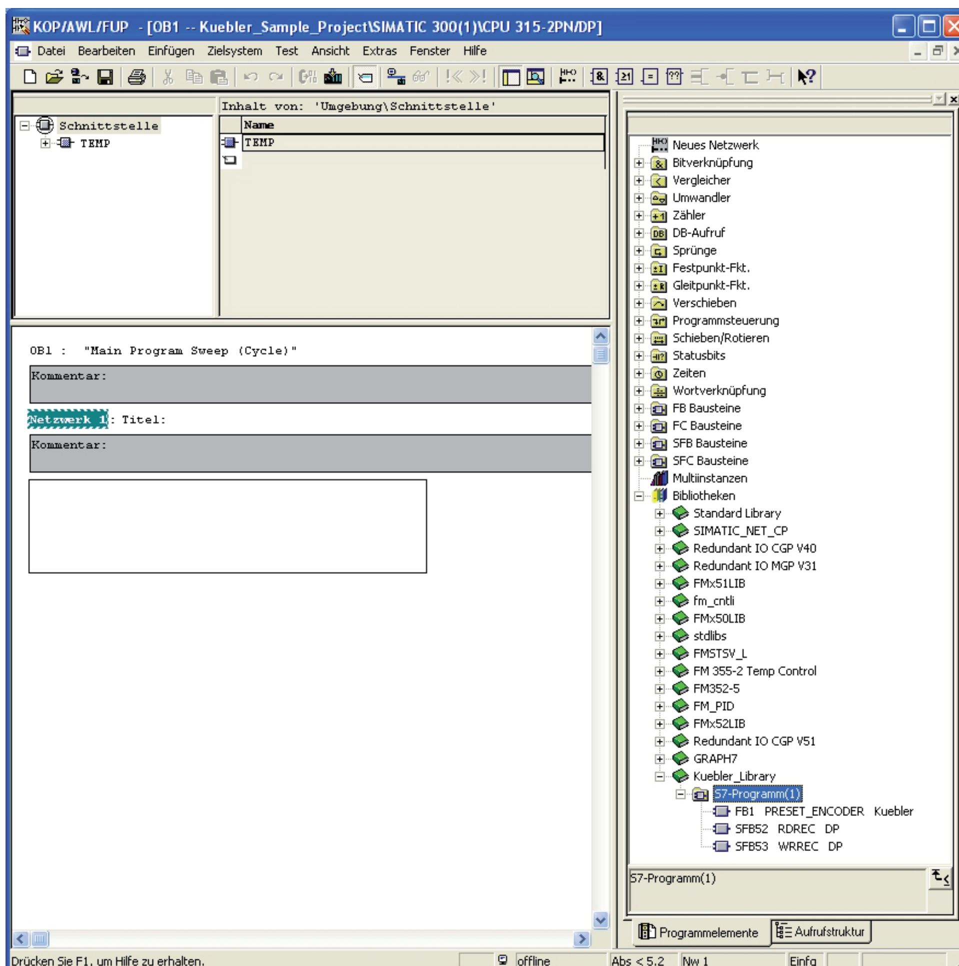


Abbildung 16

Um beim Projekt „Kuebler_Sample_Project“ zu bleiben, wird in den Abbildungen 17 bis 18 gezeigt wie dieser Baustein einzusetzen ist. Zu beachten sind hierbei die Kommentare in den Netzwerken, da sie einige Aussagen zu den Baustein-Parametern machen.

Der hier vorgestellte Bibliotheksbaustein schreibt den Preset-Wert auf den Drehgeber. Den Preset-Vorgang selbst löst er jedoch nicht aus.

Netzwerk 4: Titel:

Setzen des Preset-Wertes 0x1111 auf DG1. Gemäß HW-Konfig hat DG1 die Eingangsadresse 0, die als ID_ENCODER zu benutzen ist. Wenn der Preset-Wert erfolgreich auf den DG1 geschrieben wurde, speichern wir das Resultat auf die Variable ENC_1_SENT des OB1 um sie später für das AUSLÖSEN des Presets zu benutzen.

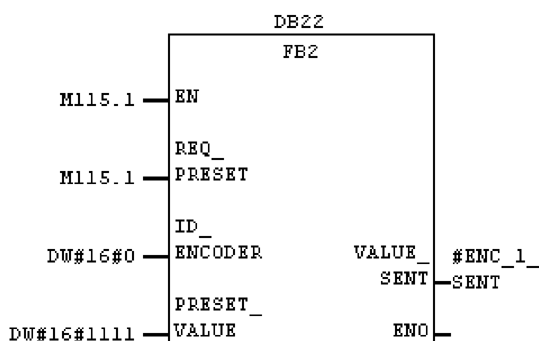


Abbildung 17

Netzwerk 6 : Titel:

Setzen des Preset-Wertes 0x1222 auf DG2. Gemäß HW-Konfig hat DG2 die Eingangsadresse 12 (0xC), die als ID_ENCODER zu benutzen ist. Wenn der Preset-Wert erfolgreich auf den DG2 geschrieben wurde, speichern wir das Resultat auf die Variable ENC_2_SENT des OB1 um sie später für das AUSLÖSEN des Presets zu benutzen.

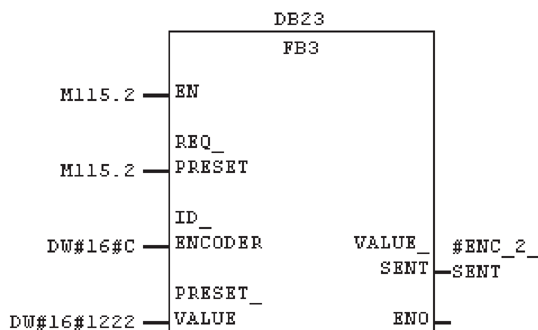


Abbildung 18

Setzen des Preset-Wertes mittels der Ezturn-Applikation
Die Ezturn-Software befindet sich im Download-Bereich des Kübler-Web-Servers und sollte über das zugehörige Setup installiert werden. Die graphische Benutzerschnittstelle erlaubt ausserdem den Zugriff auf die Ezturn-Dokumentation, die vor der Inbetriebnahme von Ezturn gelesen werden sollte.

Das Setzen des Preset-Wertes selbst ist mit Hilfe dieser Software sehr einfach und besteht aus zwei Schritten:
1. Eintrag des Preset-Wertes in die entsprechende Text-Box. Im unteren Beispiel wird der Wert 5789 gesetzt.
2. Preset-Download und Auslösen mittels des Buttons „Write preset value to encoder“. Hierbei wird der Wert nicht nur persistent (Reset-sicher) im Drehgeber gespeichert. Es wird ebenfalls der Preset-Vorgang selbst mit dem neuen Wert ausgelöst.

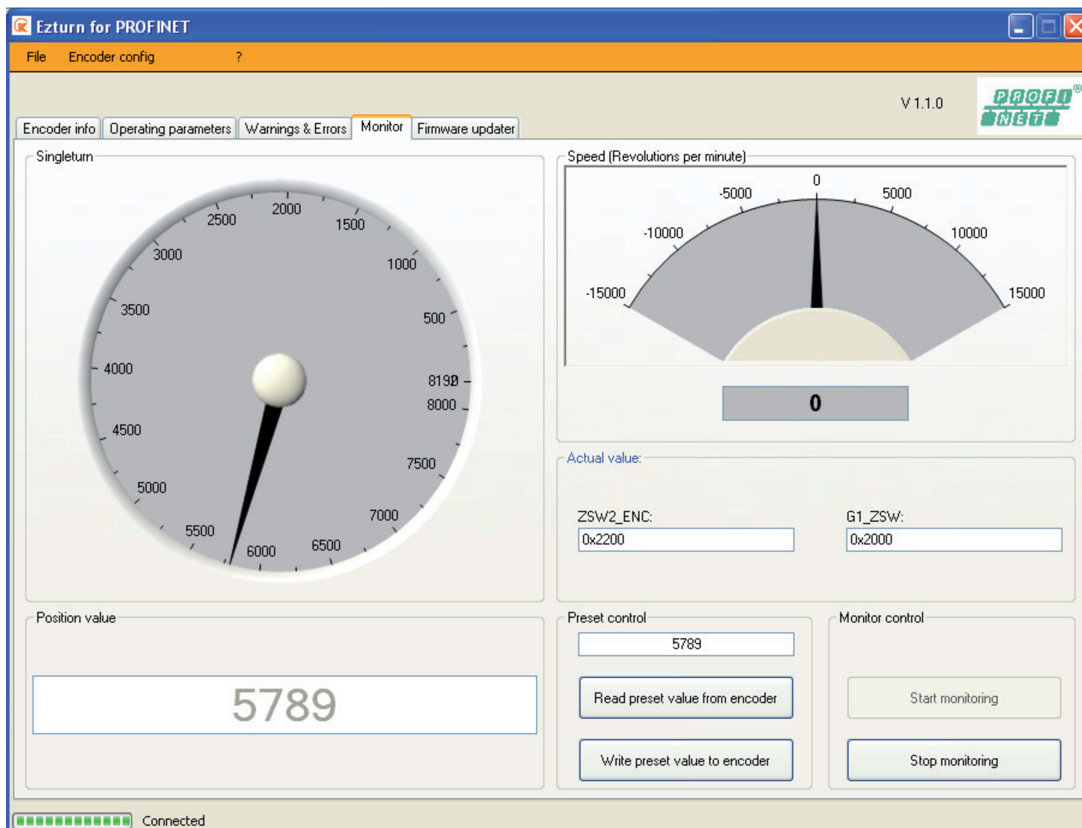


Abbildung 19: Setzen des Preset –Wertes mittels Ezturn

Setzen des Preset-Wertes mittels C-Programmiersprache
Diese Methode gilt für alle Master, die in der Programmiersprache C programmiert werden. Insbesondere also für einen CP1616-Controller, der üblicherweise in einem PC installiert und unter Linux-RTAI als Betriebssystem in Betrieb genommen wird.

Im Anhang A sind sowohl der Mechanismus in Form von UML-Sequenzdiagrammen als auch der Quellcode selbst abgebildet.

Die Drehgeber-Applikation

Die Drehgeber-Applikation ist konform zu folgenden Spezifikationen:
- Profile Encoder, Technical Specification for Profibus and PROFINET related to PROFIdrive Version 4.1 December 2008. Order No: 3.16
- Profile Drive Technology PROFIdrive Technical Specification for PROFIBUS and PROFINET Version 4.1 May 2006. Order No: 3.172.

PROFINET-MRP

Ab Firmware-Version 2.00 verfügt der Drehgeber auch über die MRP-Funktionalität (Media Redundancy Protocol). Was die Eigenschaften von MRP betrifft, so sei auf einschlägige Literatur und das Internet verwiesen.
Im Kern besteht der Vorteil von MRP darin, dass die Komponenten, die entsprechend folgender Abbildung 20 in einer Ringstruktur verkabelt sind, in ihrer Funktionalität aufrechterhalten werden wenn es zu einem Ausfall kommt oder wenn die Kabel an einer Stelle unterbrochen werden.

Im konkreten Beispiel der unteren Abbildung 20 wird die Ringstruktur durch die Steuerung zu einer Linientopologie umkonfiguriert wenn eine Unterberechung des Segments A oder C stattfindet. Der Datenaustausch mit beiden Drehgebern findet dann jeweils über den anderen Port der Steuerung statt. Die Zahlen 1 und 2 repräsentieren jeweils die Portnummer des betreffenden Gerätes.
Für den Fall, dass eine Unterbrechung des Segments B stattfindet, wird die Ringtopologie in zwei Linientopologien umkonfiguriert, an denen jeweils ein Drehgeber betrieben wird.

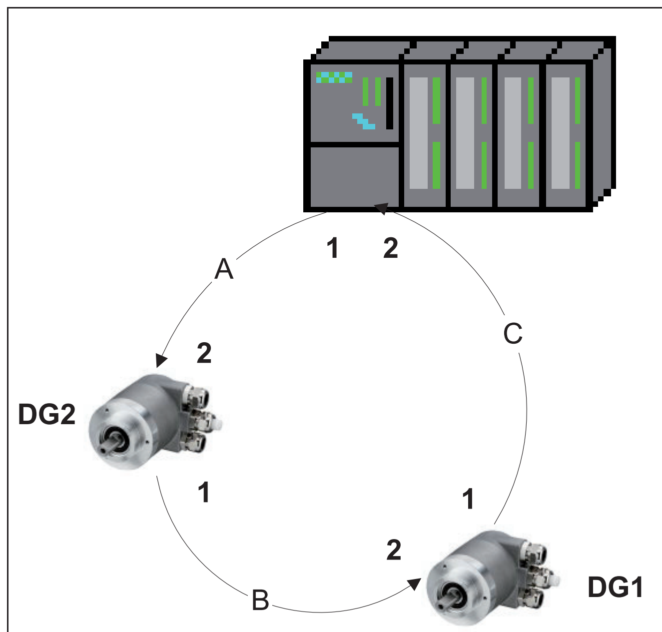


Abbildung 20

Die Konfiguration eines MRP-Projektes

Im Abschnitt „Konfiguration eines Beispielprojektes mit STEP7“ wurde aufgezeigt wie die beiden Drehgeber an der CPU315-2PN/DP betrieben werden.

Auf den folgenden Seiten wird aufgezeigt wie alle drei Komponenten für den MRP-Betrieb konfiguriert werden müssen.

Wichtig!



Für den Einsatz des MRP-Drehgebers ist die Installation der neuen GSDML-Datei GSDML-V2.2-KUEBLER-0198-Sendix58xxPNIO-20130116-131800.xml zwingend erforderlich. In der folgenden Abbildung des Hardware-Katalogs repräsentiert der mit dem roten Pfeil markierte Eintrag den MRP-fähigen Drehgeber und damit diese GSDML-Datei.

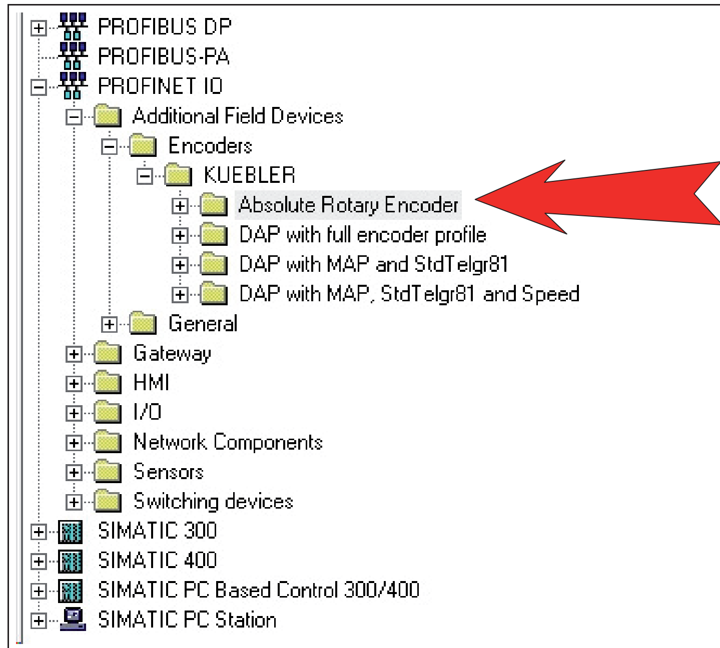


Abbildung 21

Die drei anderen Einträge darunter repräsentieren DAPs des nicht MRP-fähigen Drehgebers. Ein Doppelklick auf das Drehgeber-Symbol in der HW-Konfiguration unter STEP7 öffnet den Dialog gemäß folgender Abbildung 22, aus dem die Version der Drehgeber-Firmware und der Name der GSDML-Datei hervorgehen. Im Falle des MRP-fähigen Drehgebers muss die FW-Version mindestens 200 sein.

Properties - dg1

General | Identification

Short description: sendix58xx

Sendix 58xx PNIO

Order No./ firmware: 8.58x8.xxC2.C112 / V200

Family: KUEBLER

Device name: dg1

GSD file: GSDML-V2.2-KUEBLER-0198-Sendix58xxPNIO-20130116-131800.xml

Change Release Number...

Node in PROFINET IO System

Device number: 1 PROFINET-IO-System (100)

IP address: 192.168.0.12 Ethernet...

☑ Assign IP address via IO controller

Comment:

OK Cancel Help

Abbildung 22

Konfiguration der CPU315 2PN/DP für den MRP-Betrieb

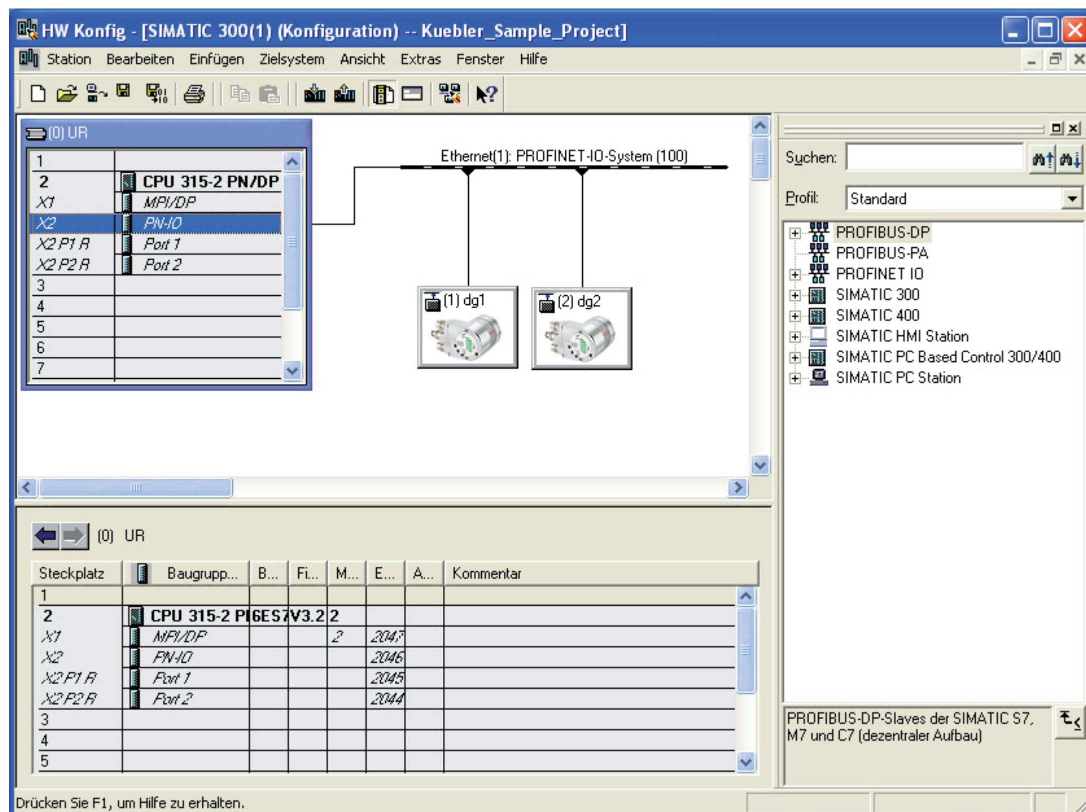


Abbildung 23

Ausgehend von Abbildung 23, auf die Zeile „PN-IO“ doppelklicken. In dem aufgehenden Dialog sind die Parameter entsprechend Abbildung 24 einzustellen. Insbesondere muss also die CPU315 als ein MRP-Manager konfiguriert werden.

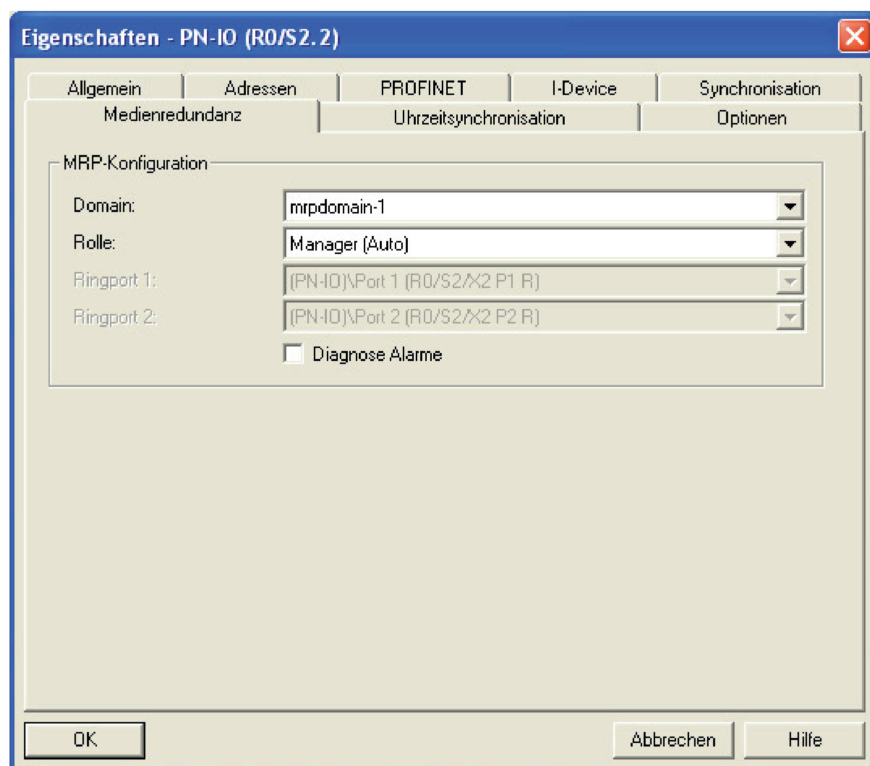


Abbildung 24

Ein Doppelklick auf die Zeile „Port1" führt zum Öffnen des Dialogs gemäß Abbildung 25, dessen Parameter wie folgt einzutragen sind. Demnach ist Port 1 der CPU315 mit dem Port 2 des DG2 verbunden.

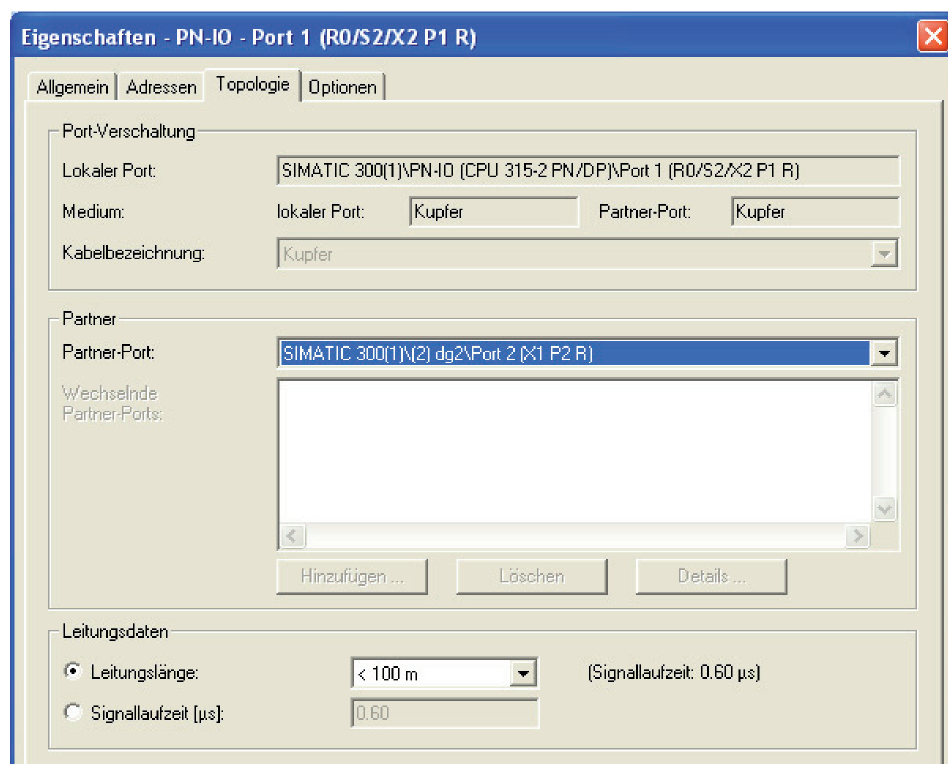


Abbildung 25

Für den Port 2 gilt der folgende Dialog, wonach Port 2 der CPU315 mit Port 1 des DG1 verbunden ist.

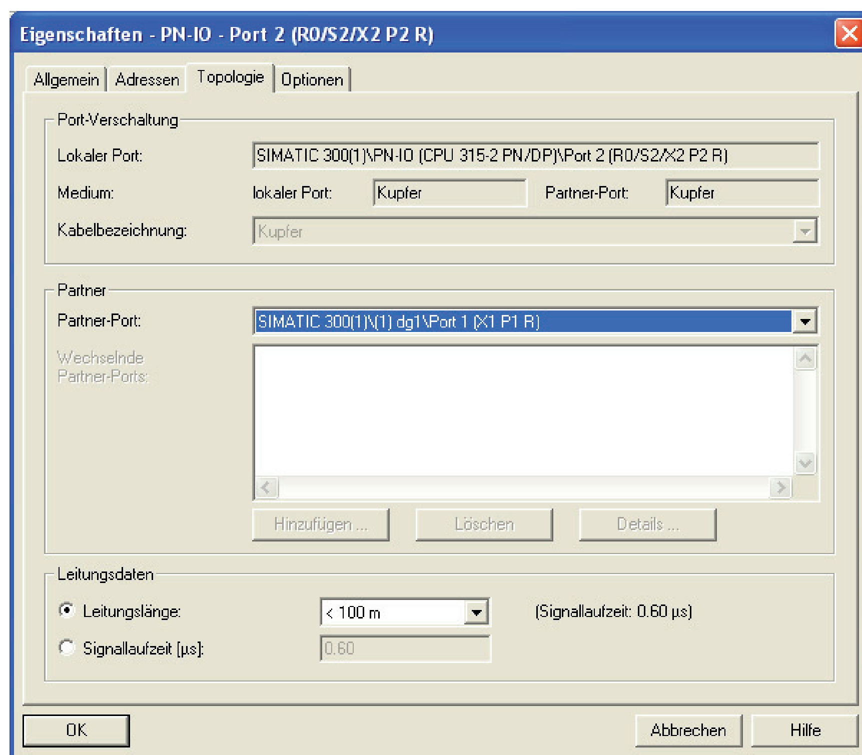


Abbildung 26

Konfiguration der beiden Drehgeber für den MRP-Betrieb
Für den DG1 gilt, ausgehend von Abbildung 27, das Analoge.

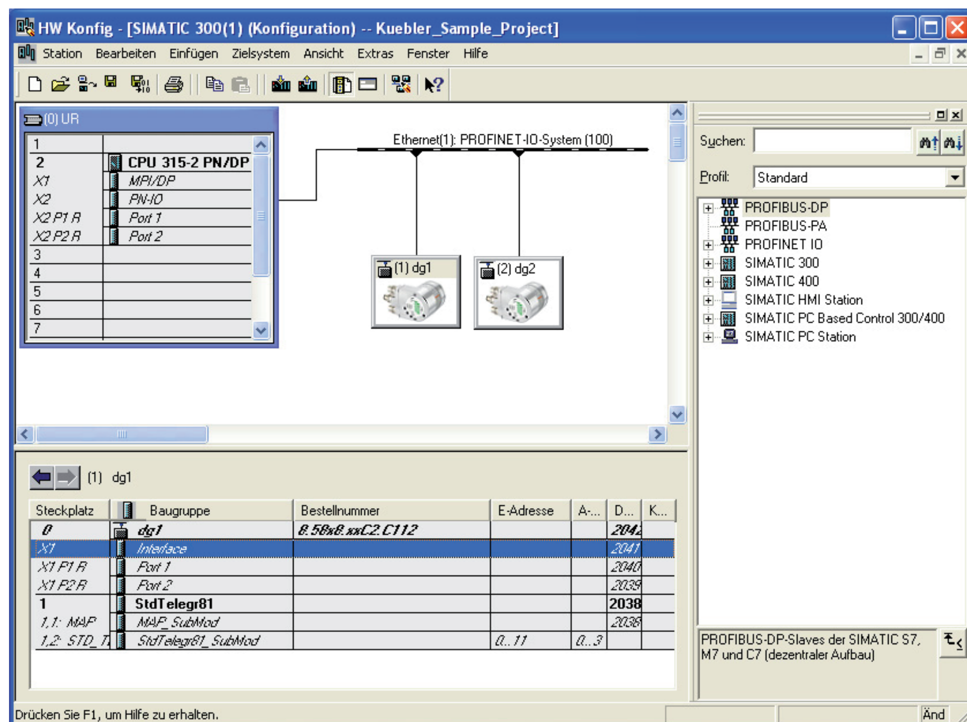


Abbildung 27

Beim Doppelklick auf Zeile „Interface“ des DG1 öffnet sich der in Abbildung 28 dargestellte Dialog. Der DG1 ist ein MRP-Client und ist daher als solcher zu parametrieren.

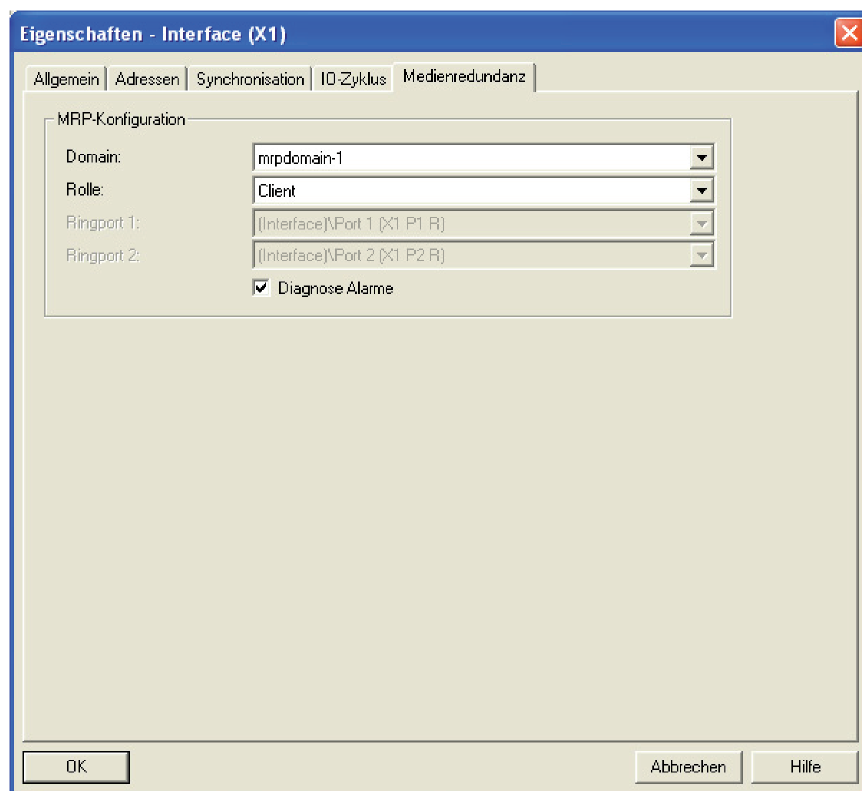


Abbildung 28

Für die beiden Ports 1 und 2 des DG1 gelten die Einstellungen aus Abbildung 29 und 30.

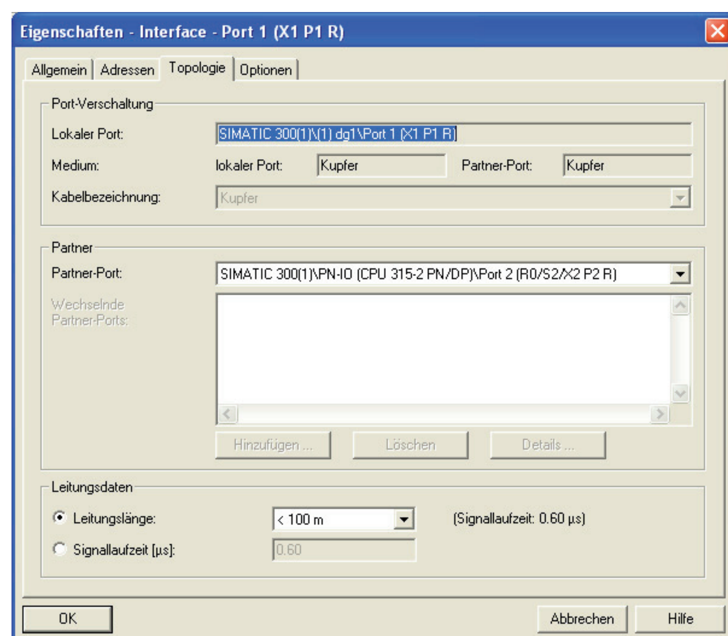


Abbildung 29

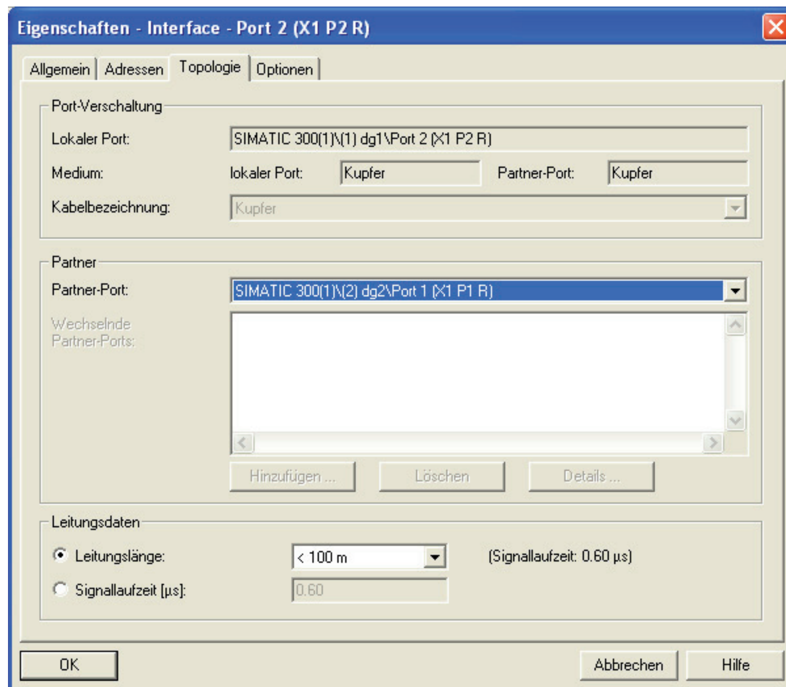


Abbildung 30

Der DG2 ist, wie DG1, als MRP-Client zu konfigurieren.

Nach einem Download über Port 1 der CPU315 wird Port 1 an den Port 2 von DG2 angeschlossen und die CPU315 über den Reset-Schalter neu gestartet.

Nach Unterbrechung des Rings, beispielsweise am Port 1 der CPU, zeigt die CPU zwar einen Fehler über die entsprechende LED an, geht jedoch nicht in den Stop-Mode über. Wird der Ring erneut geschlossen, so verschwindet auch die Fehler-Anzeige wieder.

Im Falle des unterbrochenen Rings lässt sich der STEP7 – PC anschließen und die Fehlerdiagnose wie folgt machen.

Entsprechend Abbildung 31 die Zeile „PN-IO“ der CPU315 wählen und im Menü „Zielsystem“ den Eintrag „Baugruppenzustand“ wählen.

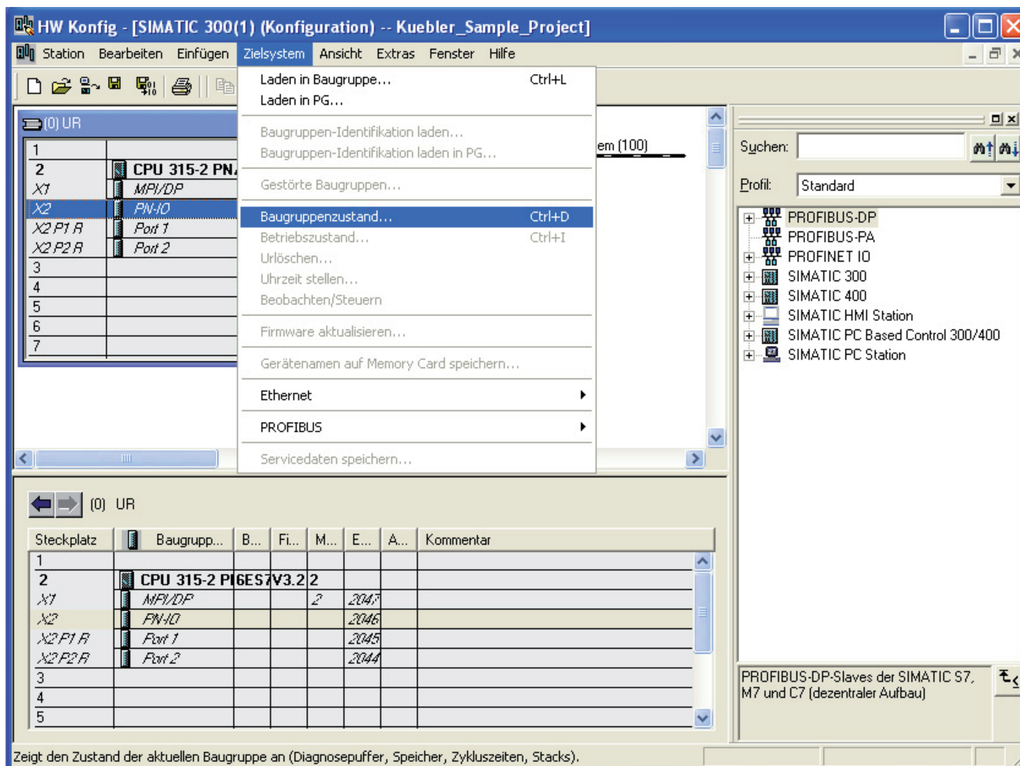


Abbildung 31

Es erscheint daraufhin ein Dialog, aus dem sich sowohl der Baugruppen- als auch der Kommunikationszustand ablesen lässt. Dies wird in den beiden Abbildungen 32 und 33 aufgezeigt.

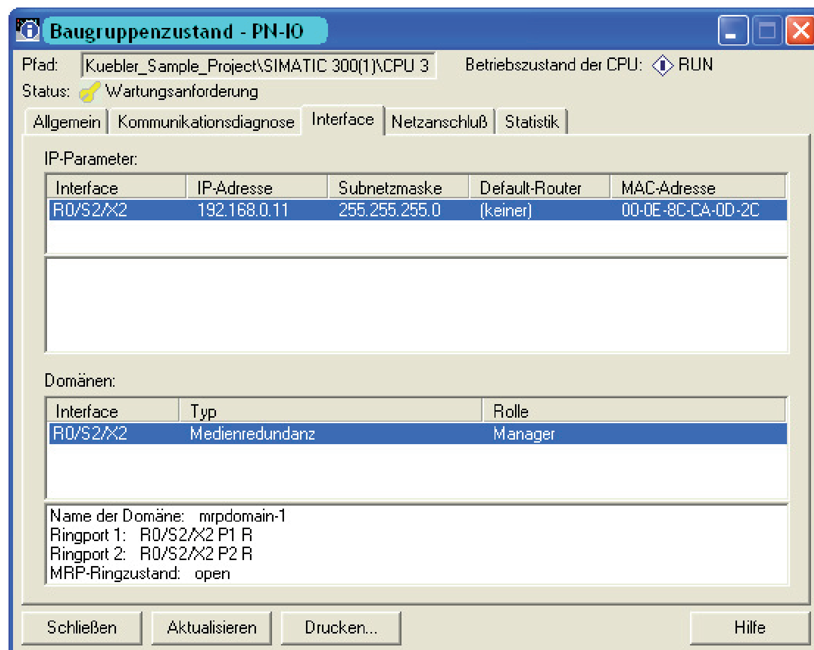


Abbildung 32

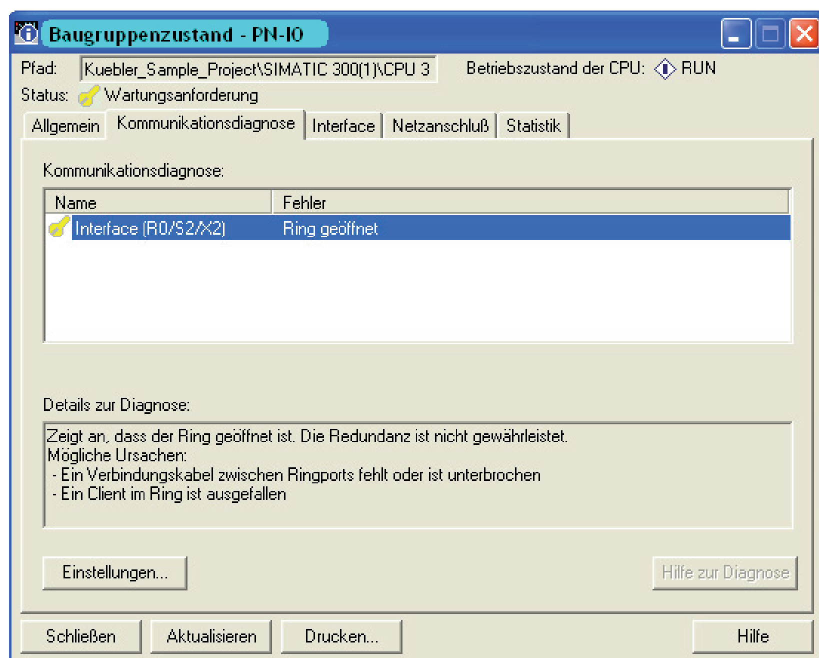


Abbildung 33

Alternativ lässt sich ein Gesamtüberblick über den Dialog verschaffen, der dann aufgeht wenn das Menü „PROFINET IO Topology..." gewählt wird. Dieser Dialog, in Abbildung 35, zeigt, dass der Port 1 der CPU und der Port 2 des DG2 einen Fehler aufweisen.

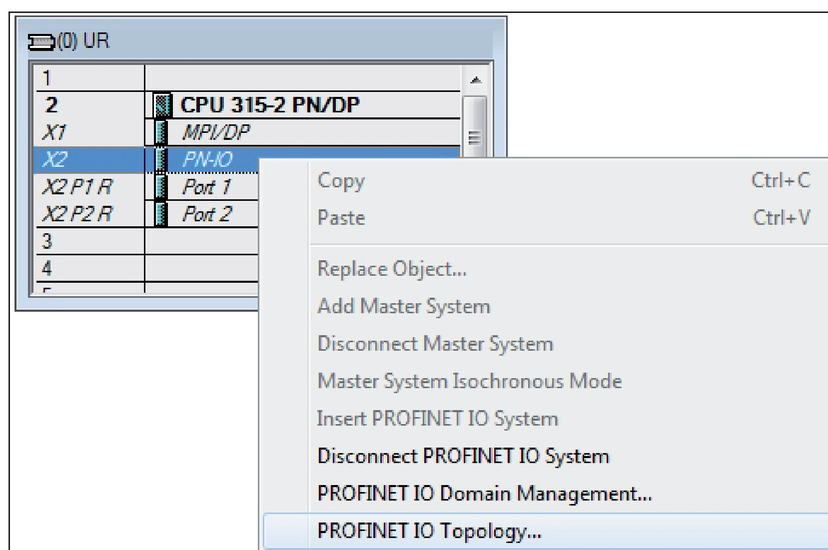


Abbildung 34

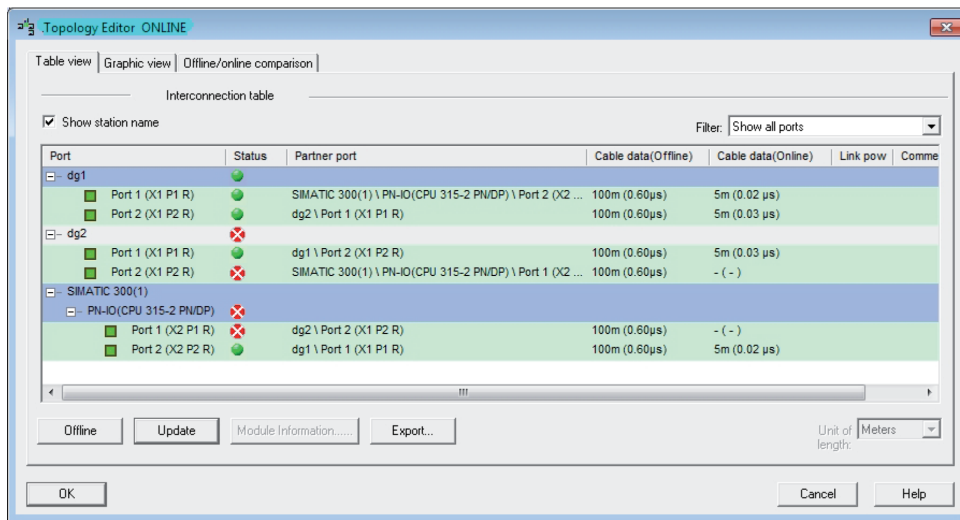


Abbildung 35

Anhang A: Prm 65000 Preset-Wert Lesen / Schreiben

Der Mechanismus zum Lesen / Schreiben des Preset – Parameters mit der Nummer 65000 ist als „Base Mode Parameter Access" (BMPA) bekannt und wird auf Seiten 59 und folgenden der „PROFIdrive profile" – Spezifikation beschrieben.

Abbildung 36 zeigt das UML-Sequenzdiagramm zum Lesen des Parameters 65000. Alle Funktionsnamen, die „kblr_" zum Präfix haben, können durch entsprechende Benutzerfunktionsnamen ersetzt werden. Alle anderen Funktionsnamen repräsentieren Funktionen aus dem PROFINET – API und können nicht ersetzt werden.

Es zeigt sich, dass der Lese-Request für den Parameter 65000 im Rahmen des BMPA-Zugriffs in Wirklichkeit ein Schreib-Request gefolgt von einem Lese-Request ist.

Die Funktion kblr_readPrm_65000_Preset füllt zunächst alle Strukturen entsprechend BMPA und übergibt sie der Funktion PNIO_rec_write_req.

Ähnlich verhält es sich bei einem Write-Request des Parameters 65000. Dieser stellt eine Kombination aus einem Write- und einem nachfolgenden Lese-Request dar.

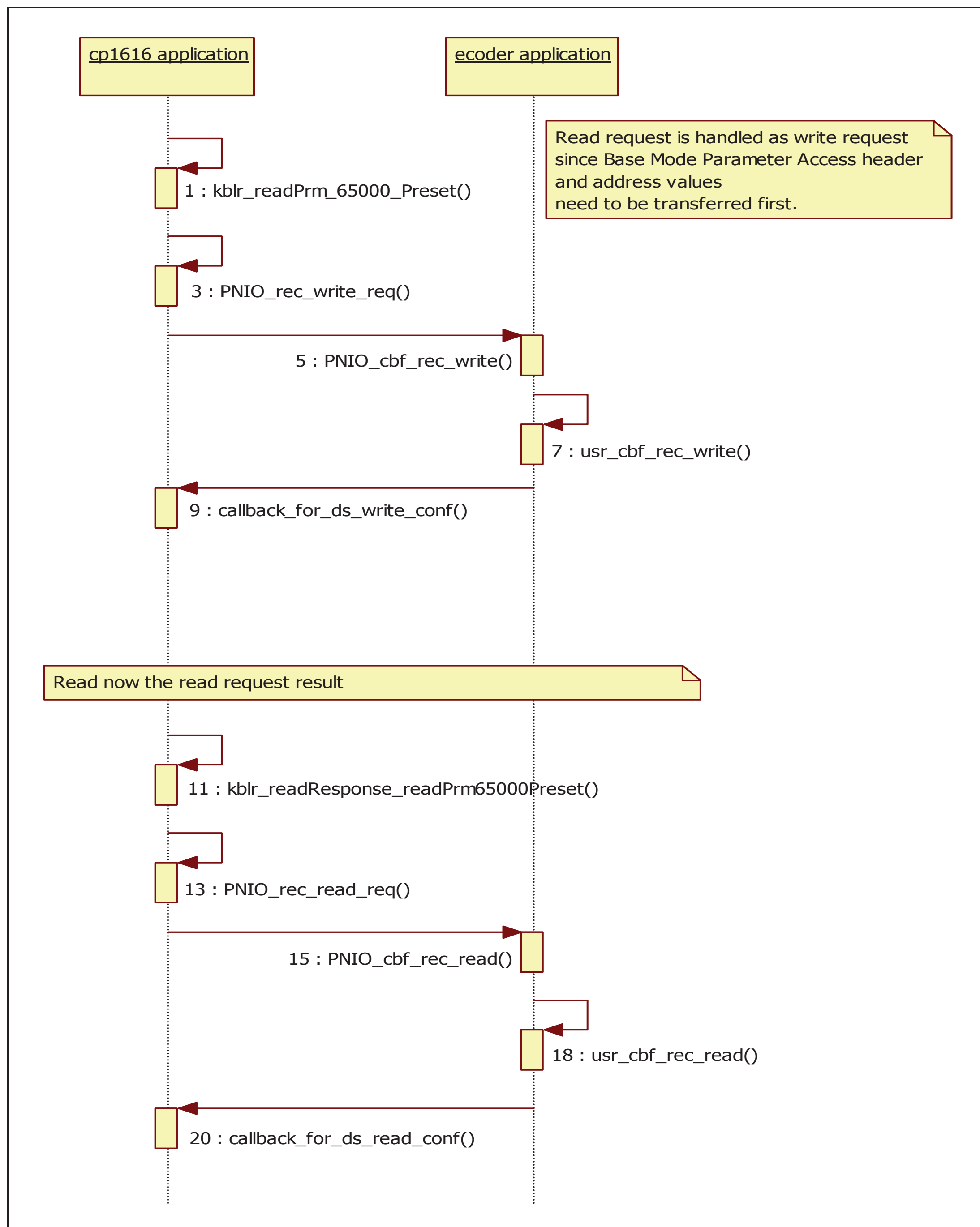


Abbildung 36

The structs are given as follows:

```
typedef struct
{
	PNIO_UINT8	RequestID;
	PNIO_UINT8	RequestRef;
	PNIO_UINT8	NoOfPrms;
	PNIO_UINT8	DO_ID;

} sBMPReqHeader;

typedef struct
{
	PNIO_UINT8	NoOfElements;
	PNIO_UINT8	Attribute;
	PNIO_UINT16	PNU;
	PNIO_UINT16	SubIdx;
} sBMPPrmAddress;

typedef struct
{
	PNIO_UINT8	NoOfValues;
	PNIO_UINT8	Format;
} sBMPPrmValue;

typedef struct
{
	PNIO_UINT8	ResponseID;
	PNIO_UINT8	RequestRefMir;
	PNIO_UINT8	NoOfPrms;
	PNIO_UINT8	DO_IDMir;
}sBMPRespHeader;

typedef struct
{
	sBMPRespHeader	BMPRespHeader;
	sBMPPrmValue	BMPPrmValue;
	PNIO_UINT8	valArray [sizeof (PNIO_UINT32)];
} sBMPResponseBuf;

typedef enum
{
	KBLR_REQUEST_PARAMETER	=	1,
	KBLR_CHANGE_PARAMETER	=	2
} KBLR_RequestID;

typedef enum
{
	KBLR_VALUE	=	0x10,
	KBLR_DESCRIPTION	=	0x20,
	KBLR_TEXT	=	0x30
} KBLR_Attribute;
```

```
typedef enum
{
      KBLR_ZERO     =  0x40,
      KBLR_BYTE     =  0x41,
      KBLR_WORD     =  0x42,
      KBLR_DWORD    =  0x43,
      KBLR_ERROR    =  0x44
} KBLR_Format;

typedef enum
{
      KBLR_REQUEST_PARAMETER_P    =   0x01,
      KBLR_CHANGE_PARAMETER_P     =   0x02,
      KBLR_REQUEST_PARAMETER_M    =   0x81,
      KBLR_CHANGE_PARAMETER_M     =   0x82
} KBLR_ResponseID;

#define KBLR_BASEMODEPRMACCESS_INDEX 0xB02E

/******************************************************************* */
/* This function acyclically reads parameter                          */
/* Prm_65000_Preset from encoder                                       */
/******************************************************************* */

void kblr_readPrm_65000_Preset (void)
{
      sBMPReqHeader       BMPReqHeader;
      sBMPPrmAddress      BMPPrmAddress;
      PNIO_UINT8*         pMem8 = NULL;

      // The logical address of the MAP/PAP submodule is 0
      // in this example
      PNIO_ADDR           SubModAddress = { PNIO_ADDR_LOG, PNIO_IO_OUT, 0 };
      PNIO_UINT32         dwErrorCode;
      PNIO_UINT32         RecordIndex = KBLR_BASEMODEPRMACCESS_INDEX;
      PNIO_REF            ReqRef = 1;

      // Now we fill the BMPA request for single value struct
      BMPReqHeader.RequestID    =   KBLR_REQUEST_PARAMETER;
      BMPReqHeader.RequestRef   =   0xAB; // To be mirrored by encoder
      BMPReqHeader.NoOfPrms     =   0x01
      BMPReqHeader.DO_ID        =   0xCD; // To be mirrored by encoder

      BMPPrmAddress.NoOfElements    =   0x00;
      BMPPrmAddress.Attribute       =   KBLR_VALUE;
      BMPPrmAddress.PNU             =   OsHtons(65000); // BIG ENDIAN !

      pMem8 = (PNIO_UINT8*)malloc(sizeof (sBMPReqHeader)+
                                  sizeof (sBMPPrmAddress));

      memcpy(pMem8,(PNIO_UINT8*)&BMPReqHeader, sizeof (sBMPReqHeader));
      memcpy(pMem8 + sizeof (sBMPReqHeader),(PNIO_UINT8*)&BMPPrmAddress,
                                  sizeof (sBMPPrmAddress));
```

```
dwErrorCode = PNIO_rec_write_req (
                        g_dwHandle,       // handle
                        &SubModAddress, // Address of the submodule
                        RecordIndex,
                        ReqRef,
                        sizeof (sBMPReqHeader) + sizeof (sBMPPrmAddress),
                        (PNIO_UINT8*)pMem8);
    free(pMem8);
}
```

Wenn callback_for_ds_write_conf ausgeführt wird, muss der Rückgabe-Wert untersucht werden um zu entscheiden ob der angeforderte Wert bereits „abgeholt“ werden kann, indem ein BMPA-Lese-Request ausgeführt wird.
Die folgende Funktion zeigt wie dies zu bewerkstelligen ist:

```
/***********************************************************************/
/* This function should immediately be called after
kblr_readPrm_65000_Preset in order to read out the result of
'parameter request' request for 65000_Preset parameter. We
/* simply do a read to index 0xB02E which is the BMPA index
/***********************************************************************/

void kblr_readResponse_readPrm65000Preset (void)
{
    // The logical address of the MAP/PAP submodule
    // is 0 in this example.
    PNIO_ADDR        SubModAddress = { PNIO_ADDR_LOG, PNIO_IO_OUT, 0 };
    PNIO_UINT32      dwErrorCode;
    PNIO_UINT32      RecordIndex = KBLR_BASEMODEPRMACCESS_INDEX;
    PNIO_REF         ReqRef = 1;
    dwErrorCode = PNIO_rec_read_req(
                     g_dwHandle,           // handle
                     &SubModAddress,       // Address of the submodule
                     RecordIndex,
                     ReqRef,
                     sizeof (sBMPResponseBuf));
}
```

Da die Funktion callback_for_ds_read_conf einen Zeiger auf die Parameterdaten und den Fehlercode liefert, ist der Kreis aus Request und Antwort geschlossen. Der folgende Text repräsentiert einen Auszug aus der Funktion callback_for_ds_read_conf.

```
Case KBLR_BASEMODEPRMACCESS_INDEX:
{
	printf („\r\ncallback_for_ds_read_conf: Receiving BMP Access data:
		0x%04x\n“, pCbfPrm->RecWriteConf.RecordIndex);

	if (pCbfPrm->RecReadConf.Err.ErrCode == 0xDE)
	{
		printf („\r\nNo BMPA mode response available yet!\r\n“);
	}
	if ( (pCbfPrm->RecReadConf.Err.ErrCode == 0) &&
		(pCbfPrm->RecReadConf.Length > 0))
	{
		for (i=0; i<pCbfPrm->RecReadConf.Length; i++)
	{
		// We simply print the received bytes
		printf („pBuf[%02d]=%02x\t“,i,
				*(pCbfPrm->RecReadConf.pBuffer+i));
		}
		printf(„\r\n“);
	}
}
break;
```

Wichtig!



Die Funktion callback_for_ds_read_conf sollte so schnell wie nur möglich zurückkehren. Zeitraubende Operationen wie printf sollten unbedingt vermieden und in Threads verschoben werden, die im niedrigeren Prioritätskontext laufen.

Abbildung 37 zeigt ein UML-Sequenzdiagramm, in dem das Szenario des Parameter 65000- Schreibens modelliert wird.

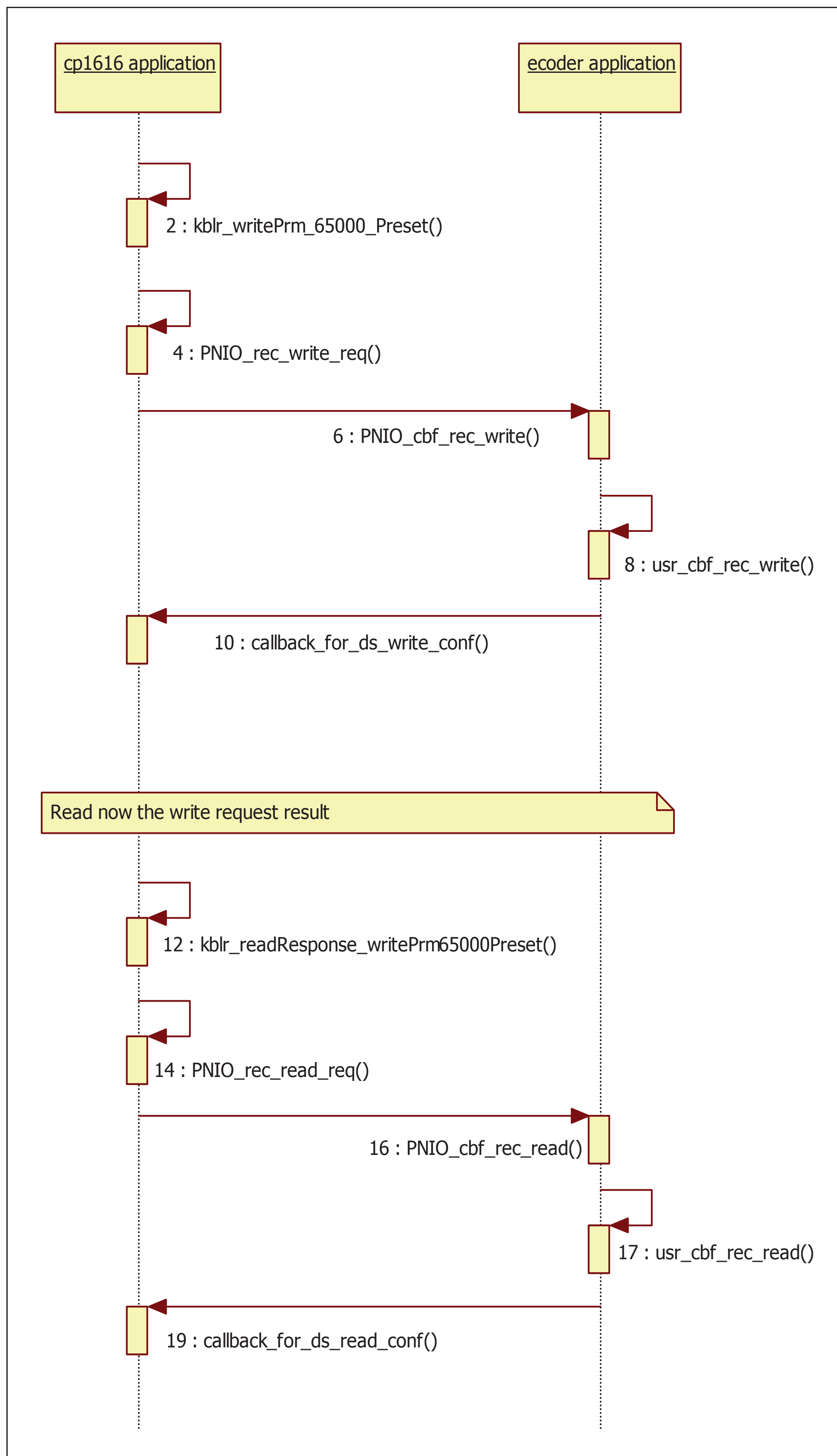


Abbildung 37

```
/***********************************************************************/
/* This function acyclically writes parameter Prm_65000_Preset to      */
   encoder                                                              */
/***********************************************************************/

void kblr_writePrm_65000_Preset (int32_t i32Preset)
{
    sBMPReqHeader       BMPReqHeader;
    sBMPPrmAddress      BMPPrmAddress;
    sBMPPrmValue        BMPPrmValue;
    PNIO_UINT8*         pMem8 = 0;

    // The logical address of the MAP/PAP submodule
    // is 0 in this example
    PNIO_ADDR           SubModAddress = { PNIO_ADDR_LOG, PNIO_IO_OUT, 0 };
    PNIO_UINT32         dwErrorCode;
    PNIO_UINT32         RecordIndex = KBLR_BASEMODEPRMACCESS_INDEX;
    PNIO_REF            ReqRef = 1;

    // Now we fill the BPMA request for single value struct
    BMPReqHeader.RequestID          = KBLR_CHANGE_PARAMETER;
    BMPReqHeader.RequestRef         = 0x12; // Should be mirrored by
                                            // encoder
    BMPReqHeader.NoOfPrms           = 0x01;
    BMPReqHeader.DO_ID              = 0x34;  // Should be mirrored by
                                            // encoder
    BMPPrmAddress.NoOfElements      = 0x01;
    BMPReqHeader.DO_ID              = 0x34; // Should be mirrored by
                                            // encoder
    BMPPrmAddress.NoOfElements      = 0x00;
    BMPPrmAddress.Attribute         = KBLR_VALUE;
    MPPrmAddress.PNU                = OsHtons(65000); // BIG ENDIAN !

    BMPPrmValue.NoOfValues          = 1;
    BMPPrmValue.Format              = KBLR_DWORD;

    i32Preset = OsHtonl(i32Preset); // BIG ENDIAN !!!

    pMem8 = (PNIO_UINT8*)malloc (sizeof (sBMPReqHeader)+
            sizeof (sBMPPrmAddress) + sizeof (sBMPPrmValue)+
            sizeof (int32_t));

    memcpy(pMem8, (PNIO_UINT8*)&BMPReqHeader,
            sizeof (sBMPReqHeader));
    memcpy(pMem8+sizeof(sBMPReqHeader),
            (PNIO_UINT8*)&BMPPrmAddress, sizeof (sBMPPrmAddress));

    memcpy(pMem8 + sizeof (sBMPReqHeader) + sizeof (sBMPPrmAddress),
            (PNIO_UINT8*)&BMPPrmValue, sizeof (sBMPPrmValue));

    memcpy(pMem8 + sizeof (sBMPReqHeader) + sizeof (sBMPPrmAddress)+
            sizeof (sBMPPrmValue), &i32Preset, sizeof (int32_t));
```

```
        dwErrorCode = PNIO_rec_write_req(
                g_dwHandle,                 // handle
                &SubModAddress, // Address of the submodule
                RecordIndex,
                ReqRef,
                sizeof (sBMPReqHeader) + sizeof (sBMPPrmAddress) +
                sizeof (sBMPPrmValue) + sizeof (int32_t),
                                            (PNIO_UINT8*)pMem8);
                free(pMem8);
        }
```

In der Funktion callback_for_ds_write_conf sollte der Rückgabewert ausgewertet und entschieden werden ob der Ergebnis-Wert des Schreib-Requests zurückgelesen werden kann.

```
/*********************************************************************/
/* This function should immediately be called after                  */
/* kblr_writePrm_65000_Preset in order to read out the result of a   */
/* parameter change request for 65000_Preset parameter.              */
/* We simply do a read to index 0xB02E which is the BMPA index       */
/*********************************************************************/

void kblr_readResponse_writePrm65000Preset (void)
{
        // The logical address of the MAP/PAP submodule
        // is 0 in this example
        PNIO_ADDR                   SubModAddress = { PNIO_ADDR_LOG, PNIO_IO_OUT, 0 };
        PNIO_UINT32 dwErrorCode;
        PNIO_UINT32 RecordIndex = KBLR_BASEMODEPRMACCESS_INDEX;
        PNIO_REF                    ReqRef = 1;
        dwErrorCode = PNIO_rec_read_req(
                g_dwHandle,             // handle
                &SubModAddress,         // Address of the submodule
                RecordIndex,
                ReqRef,
                sizeof (sBMPResponseBuf));
}
```

Der BMPA-Index Quellcode-Auszug aus callback_for_ds_read_conf trifft auch in diesem Fall zu. Allerdings sollte nicht außer Acht gelassen werden, dass wir im ersten Fall den Preset- Wert selbst gelesen haben, während wir hier das Resultat der Schreib-Operation des Preset- Wertes lesen.

Referenzen

1. PROFINET Cabling and Interconnection Technology Guideline Version 2.00 March 2007
   Order No: 2.252

2. Profile Encoder, Technical Specification for Profibus and PROFINET related to PROFIdrive Version 4.1 December 2008.
   Order No: 3.162

3. Profile Drive Technology PROFIdrive Technical Specification for PROFIBUS and PROFINET Version 4.1 May 2006.
   Order No: 3.172



Kübler Group
Fritz Kübler GmbH
Schubertstraße 47
D-78054 Villingen-Schwenningen
Deutschland
Tel.: +49 7720 3903-0
Fax: +49 7720 21564
info@kuebler.com
www.kuebler.com

wir geben Impulse

R.60713.0001_0

Kübler



Manual

Sendix 5858/5878 absolute singleturn
Sendix 5868/5888 absolute multiturn

For Order Code 8.58X8.XXCX.C2XX
From Firmware version 2.0

For Order Code 8.58X8.XXCX.C1XX
From Firmware version 1.37

pulses for automation

Liability to modification without notice
As a result of our ongoing efforts to improve our products, we reserve the right to make changes at any time to the technical information contained in the document at hand.

Warranty disclaimer
Fritz Kübler GmbH provides no guarantee, neither tacit nor express, in respect of the whole manual (whether this applies to the original German text or to the English translation) and assumes no liability for any damage, neither direct nor indirect, however caused. The specified product features and technical data shall in no case not constitute a guarantee declaration.

Document information
Revised 07/2013
Original manual. German is the original version.

Kübler Group
Fritz Kübler GmbH
Schubertstrasse 47
78054 Villingen-Schwenningen
Germany
Phone: +49 7720 3903-0
Fax: +49 7720 21564
info@kuebler.com
www.kuebler.com

Table of contents

Version of Firmware and GSDML file

Most recent version of encoder firmware and GSDML-file at release time point of this document:
Firmware version V2.00 GSDML-V2.2-KUEBLER-0198-Sendix58xxPNIO-20130116-131800.xml

Technical details and encoder features

Mechanical characteristics

Shock resistance acc. to EN 60068-2-27	2500 m/s^2, 6ms for Singleturn 2000 m/s^2, 6ms for Multiturn
Vibration resistance acc. to EN 60068-2-6	100m/s^2 , 10.....2000 Hz

Working temperature
-40…+85°C

Power supply
10…30 VDC

200 mA at 10 VDC
80 mA at 24 VDC
60 mA at 30 VDC

Hardware features
PROFINET IO ASIC: ERTEC 200
Auto-Negotiation
Auto-Polarity
Auto-Crossover
Functionality indication and diagnostic LEDs

Supported standards and protocols
RT_CLASS_1
RT_CLASS_2
RT_CLASS_3 (IRT)
DCP
RTA
LLDP
SNMP
MIB-II and LLDP-MIB
PTCP
MRP

Implemented encoder profile
Encoder Profile Version 4.1

Identification and Maintenance Functionality
Version 1.2
Supported I&M Blocks 0, 1, 2, 3, 4

Conformance with
EN 61000-4-2 :2001
EN 61000-4-3 :2006
EN 61000-4-4 :2005
EN 61000-4-5 :2007
EN 61000-4-6 :2008
EN 61000-4-7 :2004

EN 61000-6-4 :2007
EN 61000-6-2 :2006

Installation

The installation of the encoder consists of five steps:

1. Installation of data cabling
2. Installation of power cabling
3. Configuration with SIMATIC NCM PC or STEP 7
4. Installation of a PROFINET controller
5. Start of controller application along with encoders

Installation of data cabling

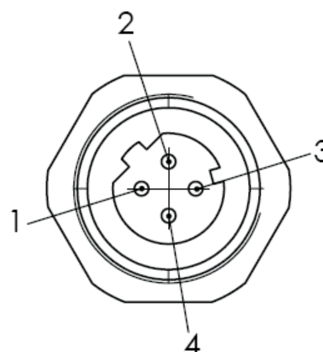
The encoder has three connectors of which two are Ethernet ports. In this documentation they will be referenced as port 1 and 2 respectively and are indicated by arrows as per sticker in picture 1 below. The middle connector is the power connector and will be described in next chapter.



Picture 1

Port 1 and port 2 connectors are 4 pin D-coded female M12 connectors. The assignment of pins is shown in picture 2 and the following table.

Assignment of signals on a female M12 connector D-coded



Picture 2 : M12 female 4 pin D-coded

Signal on M12 female 4 pin D-coded	Function	Wire colour	Pin No.
TD+	Transmit data +	Yellow	1
TD-	Transmit data -	Orange	3
RD+	Receive data +	White	2
RD-	Receive data -	Blue	4

Assignment of signals and pins for a RJ45 to M12 cable
M12 to RJ45 straight

Signal	M12	RJ45
TD+	1	1
TD-	3	2
RD+	2	3
RD-	4	6

M12 to RJ45 crossover

Signal	M12	RJ45
TD+	1	3
TD-	3	6
RD+	2	1
RD-	4	2

Recommended cable for PROFINET wiring:
Siemens Industrial Ethernet FC TP flexible Cable,
GP 2x2 (PROFINET Type B), Twisted Pair installation
Order No: 6XV1870-2B

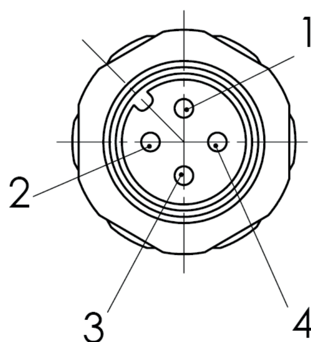
Recommended RJ45 connector:
Siemens IE FC RJ45 Order No: 6GK1901-1BB10-2AA0

IMPORTANT!



PROFINET is based on Fast Ethernet technology. Therefore a maximum segment length of 100 m only is allowed. In case of a distance bigger than 100 m, switches should be used in between. Do not use hubs! Switches have to be certified according to PROFINET specifications. If the encoder is configured for MRP (Media Redundancy Protocol) operation, the switches inserted in between must be managed and, in case of 60 IRT mode, they must also be IRT-capable! Example of a switch that is both managed and IRT-capable: SCALANCE 200 IRT.

Installation of power cabling
Picture 3 and the following table depict pin assignment of the 4 pin A-coded male power connector on encoder.



Picture 3: M12 male 4 pin A-coded

Signal on M12 male 4 pin A-coded	Function	Pin No.
PWR	10 – 30 V DC	1
		2
GND	0V	3
		4

For more details on cabling in PROFINET environments, see specification PN-Cabling-Guide_2252_V200_May07.pdf.

This specification can be downloaded from the non-member area at
http://www.profibus.com/downloads/

Diagnostic LEDs

The encoder has four diagnostic LEDs which are labelled as shown in the following picture.



Picture 4

LED label	Colour	Functionality description
LINK 1	Yellow and green	LINK 1 is a bicolour LED for port 1 which indicates link activity (green) and data transfer activity (yellow)
LINK 2	Yellow and green	LINK 2 is a bicolour LED for port 2 which indicates link activity (green) and data transfer activity (yellow)

The following table describes all operating situations indicated by combination of ERROR- Led and PWR-Led.

ERROR LED (red)	PWR LED (green)	Meaning	Possible cause
OFF	ON	Normal operation. Data exchange ok	
Blinking	ON	Bus data exchange possible, however encoder did not switch to process data exchange mode. Encoder indicates error number in G1_XIST2. Alternatively the error code can be read out by reading parameter 65001	See error blink codes in next chapter
ON	ON	Bus data exchange possible, however no data exchange on bus	Master not available or bus disconnected
OFF	OFF	No power	

Error blink codes

Blink Code	Cause
Once every 2 seconds (0.5Hz) Error LED ON for 1 sec., OFF for 1 sec., then repeated	- Slave not yet con figured by controller. The User Parameter Data in terms of 0xBF00 index data set has not been received yet by the encoder. - Wrong configuration - Wrong station address assigned (but not outside the permitted range) - Actual configuration of the slave differs from the nominal configuration
5 times per second Error LED ON for 0.1 sec., OFF for 0.1 sec., then repeated	Bus communication ok, however encoder object has no connectivity to position data sensor
Once per second (1Hz) Error LED ON for 0.5 sec., OFF for 0.5 sec., then repeated	Memory error

Sample project configuration with STEP 7

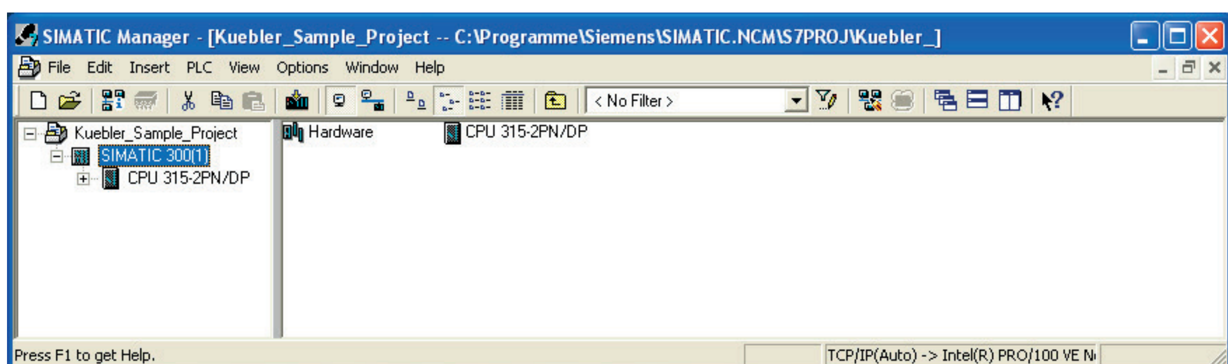
Important!



For project configuration, use imperatively STEP7 version V5.5. Otherwise, errors will occur during the installation of the GSDML file of the encoder and of all other GSDML files that use the same XML schema. In addition, errors will occur when configuring and parameterising MRP.

There is a sample STEP 7 project called “Kuebler_Sample_Project” ready to be downloaded from Kübler web server. This project will be discussed on next pages. There is also the GSDML file of the MRP-capable PROFINET encoder which must be installed before starting up the encoder under STEP7.

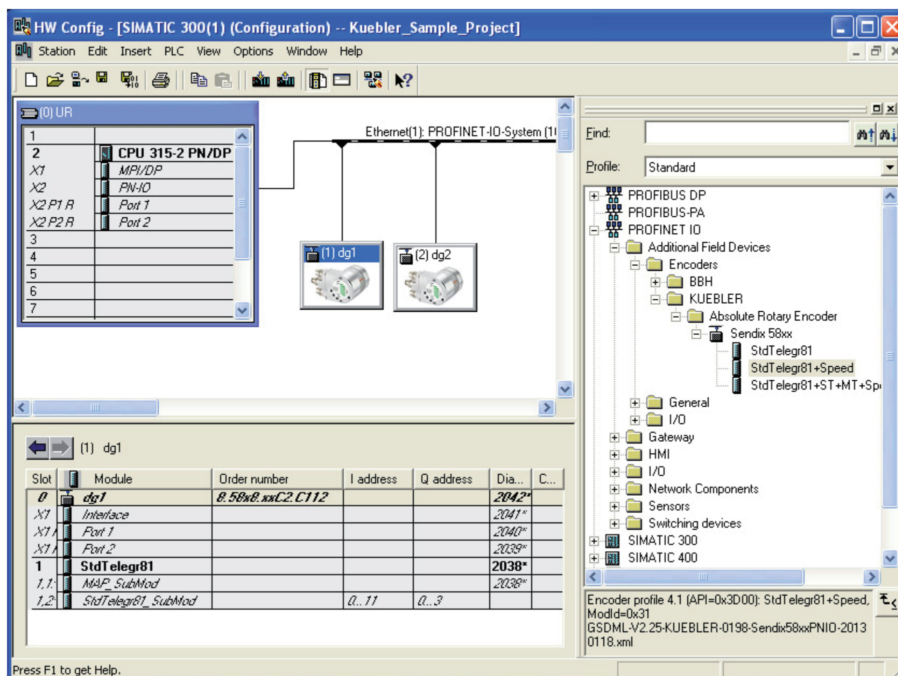
The two following pictures represent the project itself and the hardware configuration with two encoders, which all use the same DAP, however with different modules.



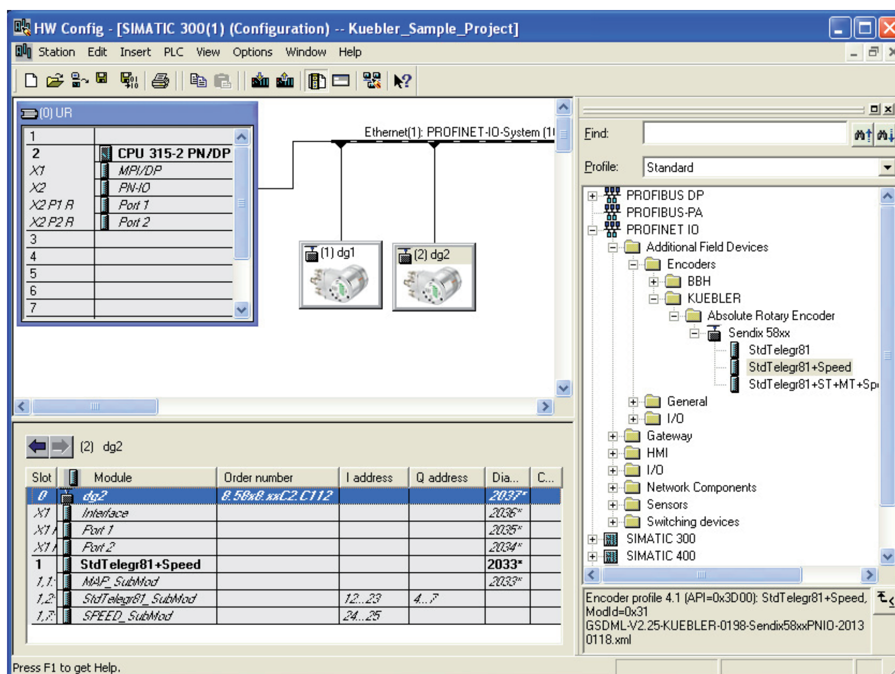
Picture 5

The first encoder dg1 uses the module with standard telegram 81, represented in picture 6. Its address for reading standard telegram 81 is 0. The write address for standard telegram 81, with which, among others, the preset is triggered, is 0 as well.

The second encoder dg2 uses the module that includes both standard telegram 81 and the speed. The two addresses for reading and writing standard telegram 81 are 12 and 4, as shown in picture 7.



Picture 6

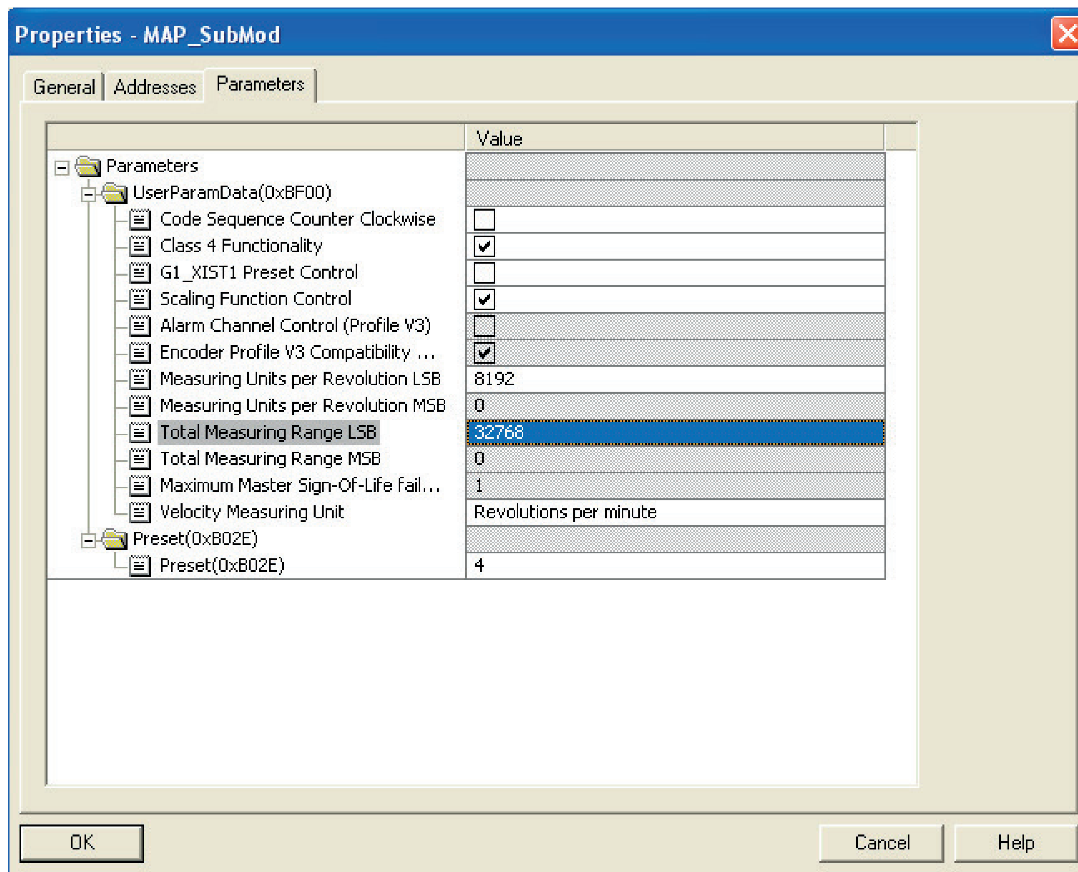


Picture 7

Configuration of the encoder user parameters

The user parameter data is defined on page 52 of the encoder profile specification and can be sent as a record data object to the encoder during its startup phase. Members of the record data object are depicted in the following dialog screenshot.

In order to start the dialog for parameter configuration, double click in the hardware configuration of the respective encoder on line "MAP_SubMod".



Picture 8

The parameter values are passed from the PLC to the encoder during the preparation phase of the encoder base model state machine.

The dialog is to be started by a double click on row entitled "MAP-SubMod" which is slot 1 subslot 1 of the encoder configuration dialog. Since the Sendix type of encoder is a 16 bit single turn resolution and a 12 bit multi turn resolution encoder, the corresponding 4 most significant bytes of Measuring Units per Revolution and Total Measuring Range are not changeable. Consequently they have a grey background colour and value zero per default.

Values in picture 10 are automatically checked when user clicks button OK. If necessary STEP7 displays the proper value range and prompts user to correct his values.

For MUR and TMR the value range is derived from following facts:

- The MUR value (Measuring Units per Revolution) will be only accepted if it matches following criteria:
  0 < MUR ≤ g_ST
  where g_ST is the physical single turn resolution (65536 according to 16Bit).

- For an encoder **without** multi turn unit the TMR value (total measuring range) will be accepted only if it matches following criteria:
  0 <TMR ≤ g_ST
  where g_ST is the physical single turn resolution which is 65536 (16 Bit).

- For an encoder **with** multi turn unit the following criteria must match:
  0 < TMR ≤ MUR*g_MT
  where MUR is the Measuring Units per Revolution value and g_MT the physical multi turn resolution (4096 according to 12Bit) in case of the Sendix type described herein.

Caution!

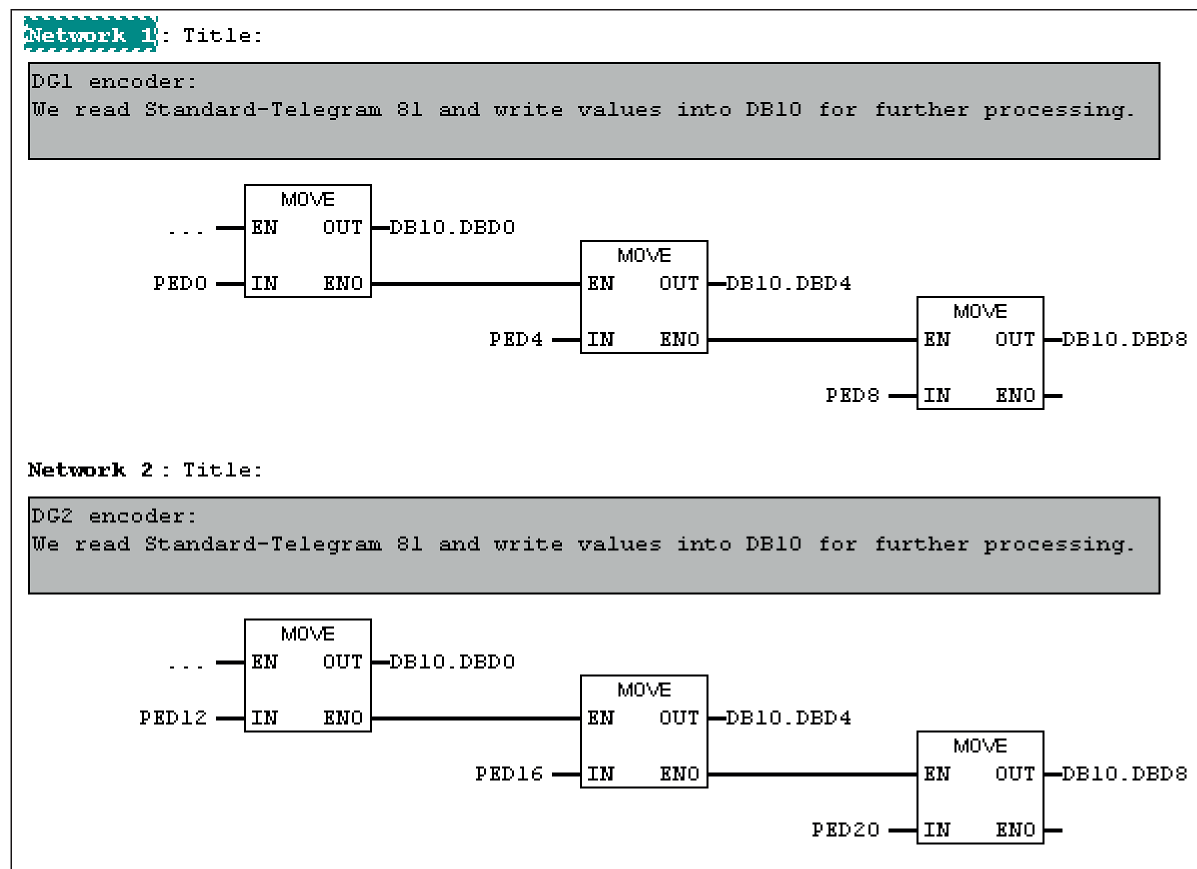


The single turn and multi turn resolution values should be chosen as values given by 2^X (2 to power X). In this case there is no remaining part smaller then single turn value when multiturn reaches its maximal value given by the maximal physical multiturn resolution.

For the Preset-Value following criteria apply:
- 0 ≤ Preset ≤ TMR-Wert AND
- Preset ≤ 0x7FFFFFFF (MaxINT)

Reading of encoder position values

The functionality of the OB1 block has been implemented in order to show in a very simple way the principle of position value reading. In following picture 9, the two networks reflect this mechanism. Here, starting at addresses 0 and 12, three double words, which correspond to the length of standard telegram 81, are read and stored in DB10 for further processing.

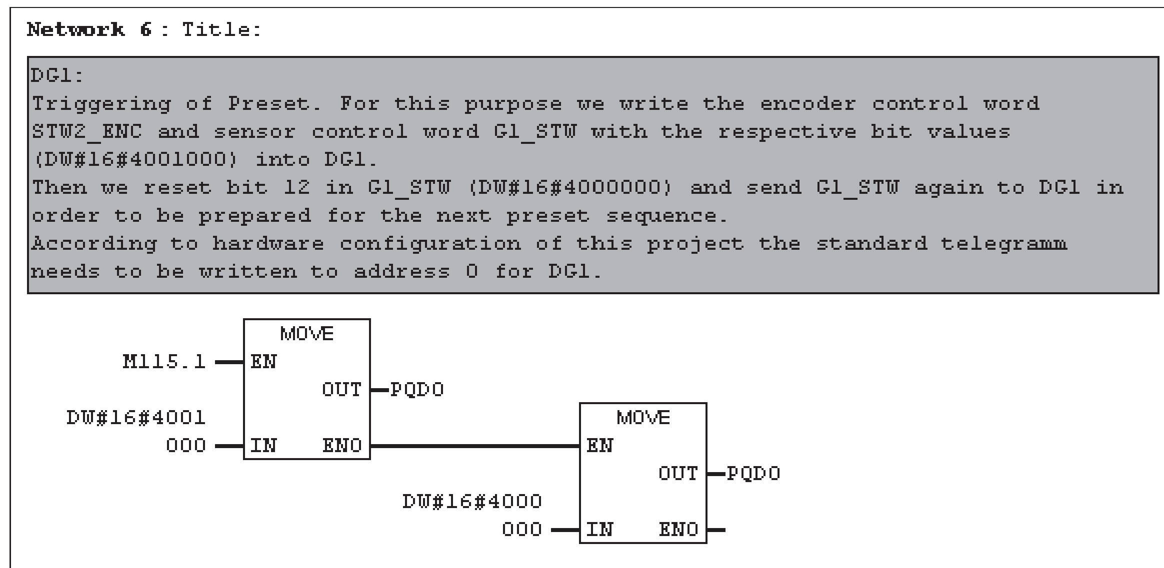


Picture 9: Implementation of OB1. Reading of position values.

Triggering of Preset

Within the PROFINET encoder profile the preset value is also known as parameter 65000. Per default the preset value is zero but can be changed according to dialog in picture 10. The prerequisites for doing this were discussed in chapter "configuration of the encoder user parameters".

The following picture 10 shows the network which is to be used in order to trigger an absolute preset on encoder dg1.

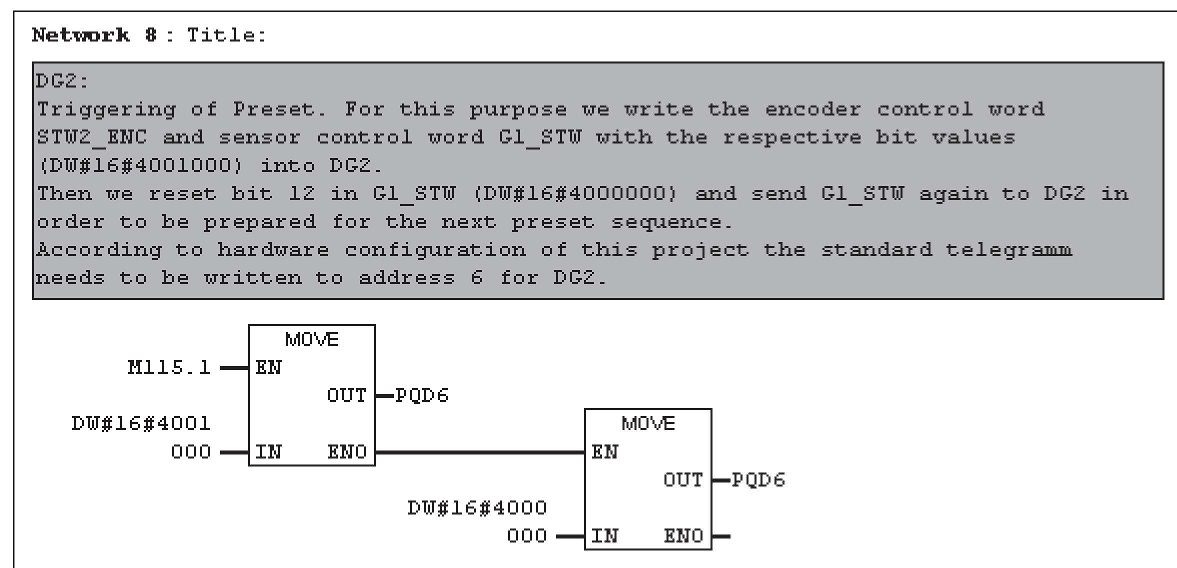


Picture 10: Triggering of preset on dg1

In order to trigger an absolute preset on encoder, the encoder control word STW2_ENC with PLC bit in position 10 set and sensor control word G1_STW with bit in position 12 set needs to be sent to the encoder.

To be able to retrigger a preset, the encoder has to receive G1_STW with bit 12 reset.

For encoder 2 picture 11 applies.

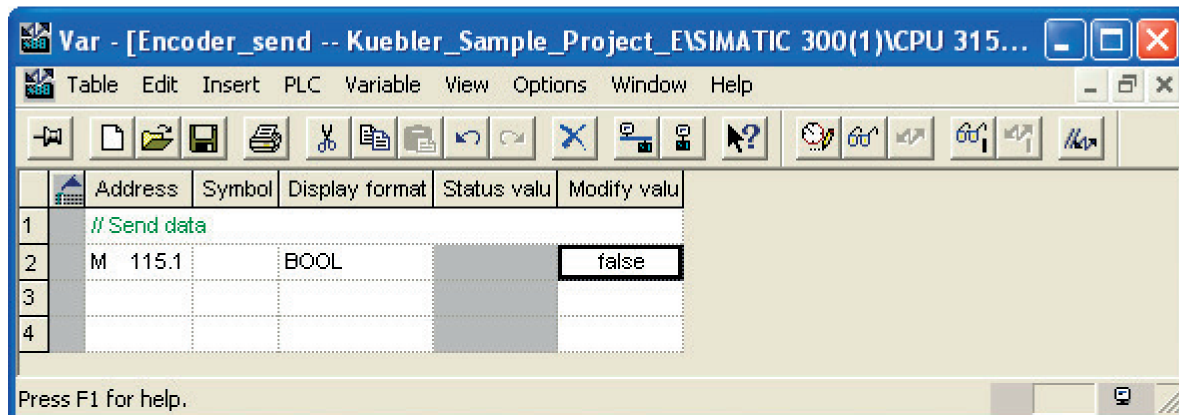


Picture 11: Triggering of preset on dg2

The triggering of the preset itself is achieved in this example by setting bit 1 of M115. This is shown in picture 12 in the form of a variable table. In case of an online connection between the PC and the PLC this bit can be set and reset.

For more details on the encoder see reference [2].

The purpose of using a variable table in this particular case is only to show the principle of how it works. In the field there might be real (hardware) switches which control the action of triggering a preset.



Picture 12

Download of parameter 65000 Preset

In the previous chapter we emphasized the triggering of preset. In this chapter we are going to emphasize on the download of the preset parameter value itself.

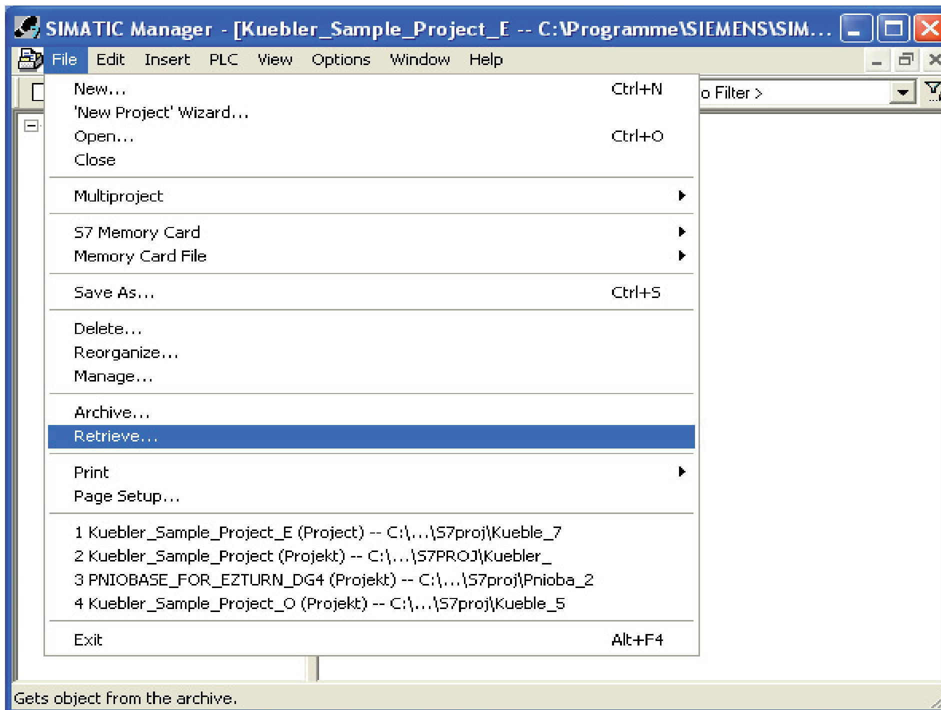
The preset value at download time is only accepted if it matches the following criteria:

0 ≤ Preset ≤ TMR (SubIdx 10 Parameter 65001) and Preset ≤ MaxINT32

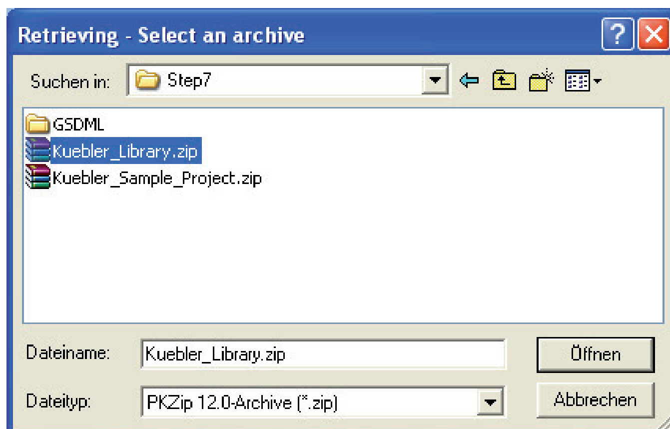
According to the specification, the preset value is based on scaled units, hence the Operating Status control bit “Scaling function control” must be set as well as “Class 4 functionality”. Also both values Measuring Units per revolution (MUR) and TMR (Total measuring range) must be set properly in advance.

In picture 8 was shown how to send the preset value to the encoder on startup time of the system.
There are three more ways to send the preset value to the encoder. They will be shown in the next chapters.

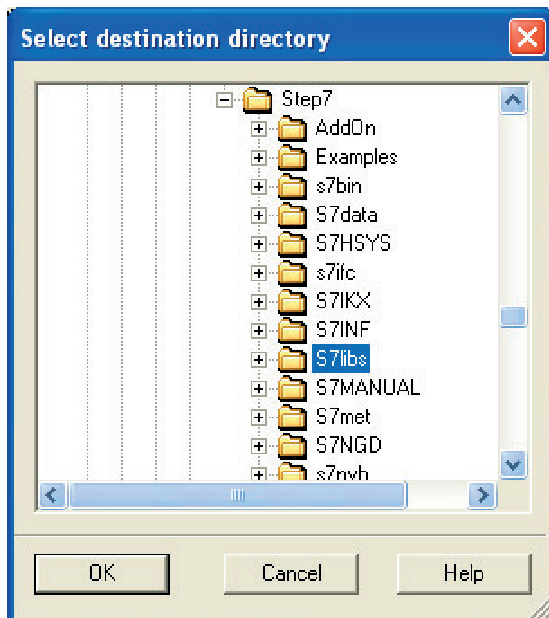
Download of the preset value using Kuebler-FB1-STEP7 Block
There is a Kuebler specific library called “Kuebler_Library.zip” on our web server. The function block FB1 is capable of sending the preset value to the encoder at any time out of the PLC program.
The installation of the Kuebler library works according to picture 13 up to 15. That means in particular that the library gets retrieved and unzipped in the directory S7libs of the STEP7- Installation directory.



Picture 13

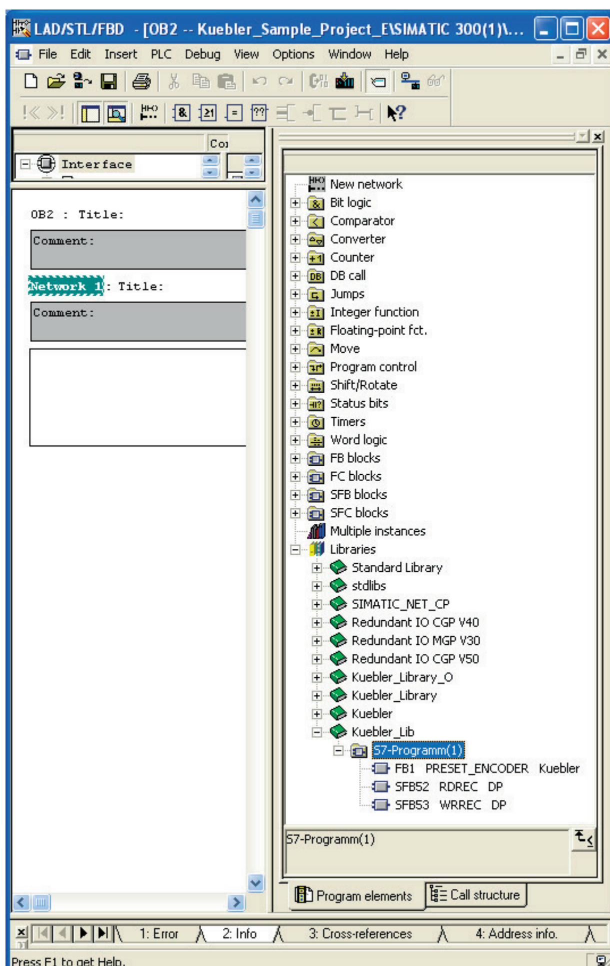


Picture 14



Picture 15

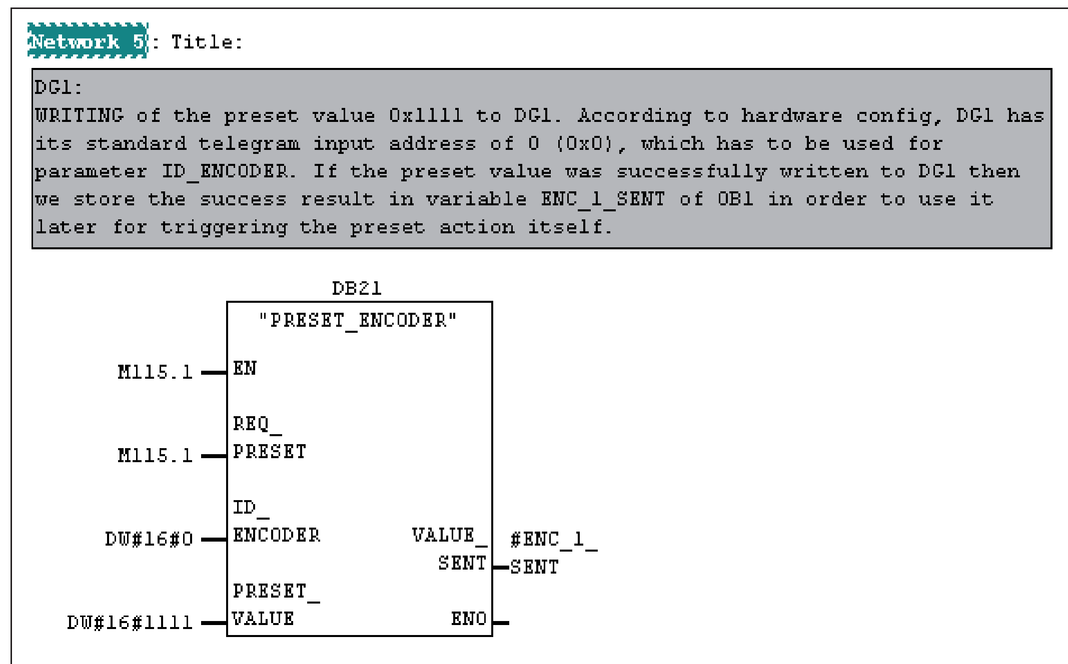
The library with its components becomes visible within the block editor on the program elements tab when retrieving of the library succeeded. This is shown in picture 16. From now on the function block “FB1 PRESET_ENCODER Kuebler” can be used per drag&drop in any other blocks created by the programmer.



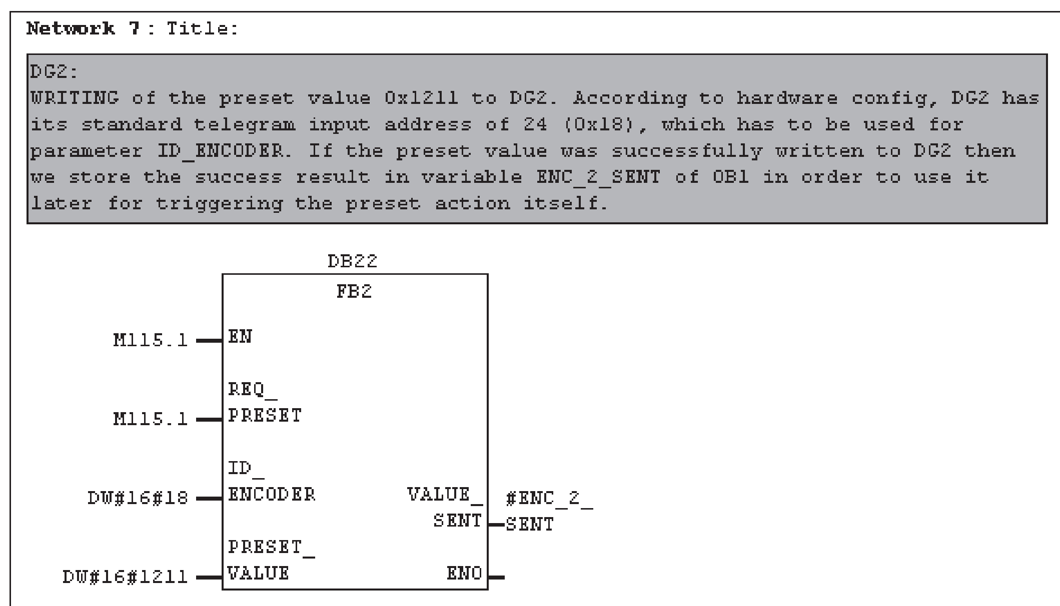
Picture 16

In order to stay with the project “Kuebler_Sample_Project”, pictures 17 up to 18 demonstrate how to use the Kuebler library function block. Important in all four pictures are the comments given in the network implementations itself. In particular they explain the block input and output parameters.

It is important to keep in mind that the Kuebler library function block only downloads the preset value but does not trigger the preset action itself.



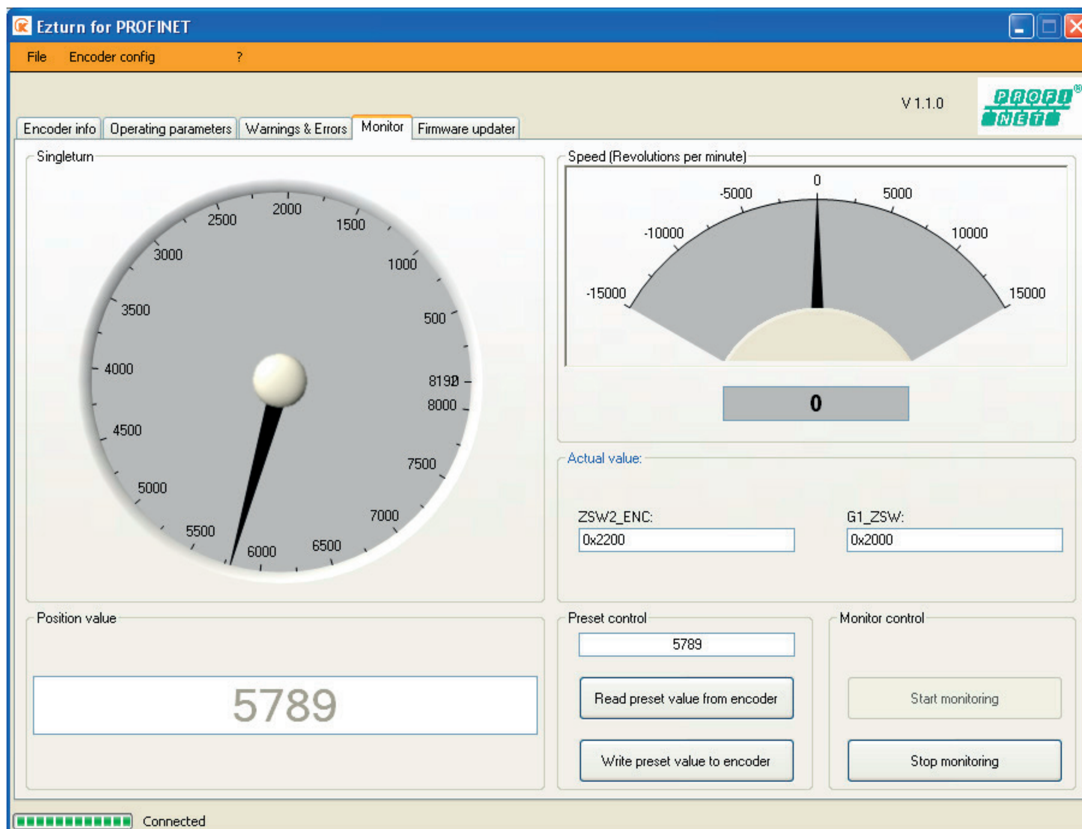
Picture 17



Picture 18

Writing of the preset value by using the Ezturn-Application
The Ezturn Software along with its documentation can be downloaded from our web server. The setup program is located in directory "Ezturn CANopen_ProfiNet_RS485". Double click on CDStart.exe in order to start the installation.

Writing of the preset value to the encoder is very simple by using Ezturn and consists out of two steps.
1. In the group box "Preset control" of the monitor tab write the preset value into the edit field. This is value 5789 in the example shown below.
2. Press button "Write preset value to encoder" which leads to persistant storage of the preset value (reset save) AND triggers the preset action itself.



Picture 19 : Write and activate preset by Ezturn application

Writing of the preset value by using C language programming
This method applies for all masters which are programmed in C language. This particularly applies for CP1616 controller based masters which are installed in a PC and run under Linux – RTAI.

Appendix A covers this method by showing the respective UML-sequence diagrams as well as the source code itself.

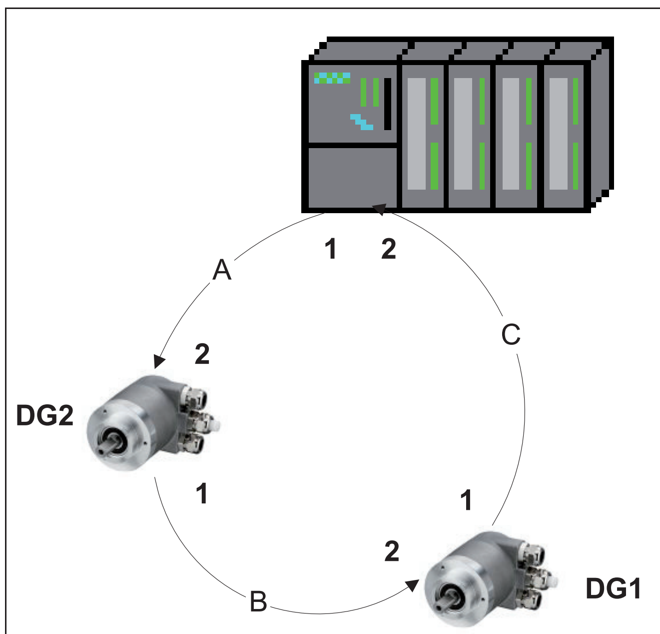
The encoder application

The encoder application conforms with following specifications:
- Profile Encoder, Technical Specification for Profibus and PROFINET related to PROFIdrive Version 4.1 December 2008. Order No: 3.162
- Profile Drive Technology PROFIdrive Technical Specification for PROFIBUS and PROFINET Version 4.1 May 2006. Order No: 3.172.

PROFINET-MRP

As from firmware version 2.00; the encoder also includes the MRP (Media Redundancy Protocol) functionality. For further information about the MRP features, please refer to the relevant literature and to the Internet.
Basically, the advantage of MRP is that the functionality of the components, which are wired in a ring structure as shown in picture below, is maintained in case of a failure or of a breakage of the wires in any location.

In the concrete example of picture 20 below, the ring structure is reconfigured by the control into a line topology in case of the breakage of segment A or C. Data exchange with both encoders then takes place respectively via the other port of the control. The pictures 1 and 2 represent the respective port numbers of the concerned device.
In case of the breakage of segment B, the ring topology is reconfigured into two line topologies, each of them controlling one encoder.



Picture 20

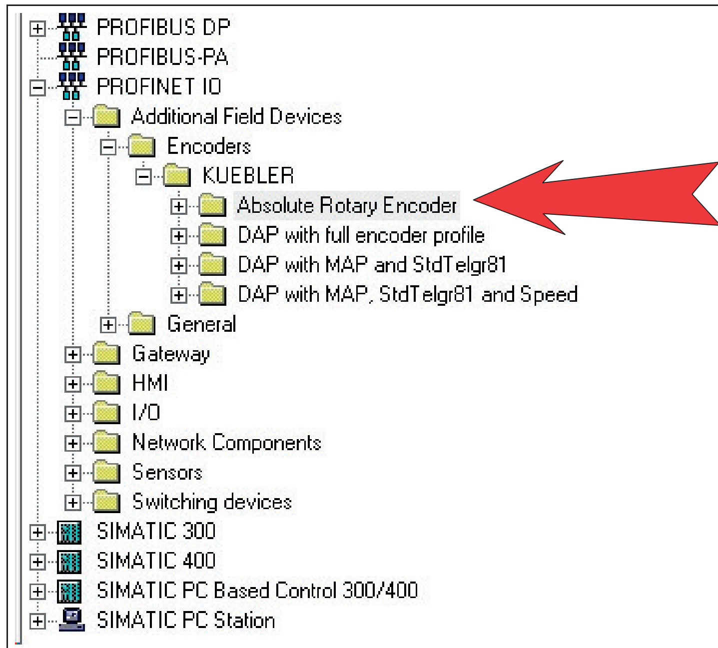
Configuration of an MRP project

Section “Sample project configuration with STEP7” shows how both encoders are operated on CPU315-2PN/DP.

The following pages show how to configure all three components for MRP operation.

Important!

The operation of the MRP encoder requires imperatively the installation of the new GSDML file GSDML-V2.2-KUEB-LER-0198-Sendix 58xxPNIO-20130116-131800.xml. On the following picture representing the hardware catalogue, the entry shown by the red arrow represents the MRP-capable encoder, and thus this GSDML file.



Picture 21

The three other entries below the one shown by the arrow represent DAPs of the not MRP- capable encoder. Double-clicking on the encoder icon in the HW configuration under STEP7 opens the dialogue window represented in picture 22 below, which shows the encoder firmware version and the name of the GSDML file. For the MRP-capable encoder, the FW version must be at least 200.

Properties - dg1

General | Identification

Short description: sendix58xx

Sendix 58xx PNIO

Order No./firmware: 8.58x8.xxC2.C112 / V200

Family: KUEBLER

Device name: dg1

GSD file: GSDML-V2.2-KUEBLER-0198-Sendix58xxPNIO-20130116-131800.xml

Change Release Number...

Node in PROFINET IO System

Device number: 1 PROFINET-IO-System (100)

IP address: 192.168.0.12 Ethernet...

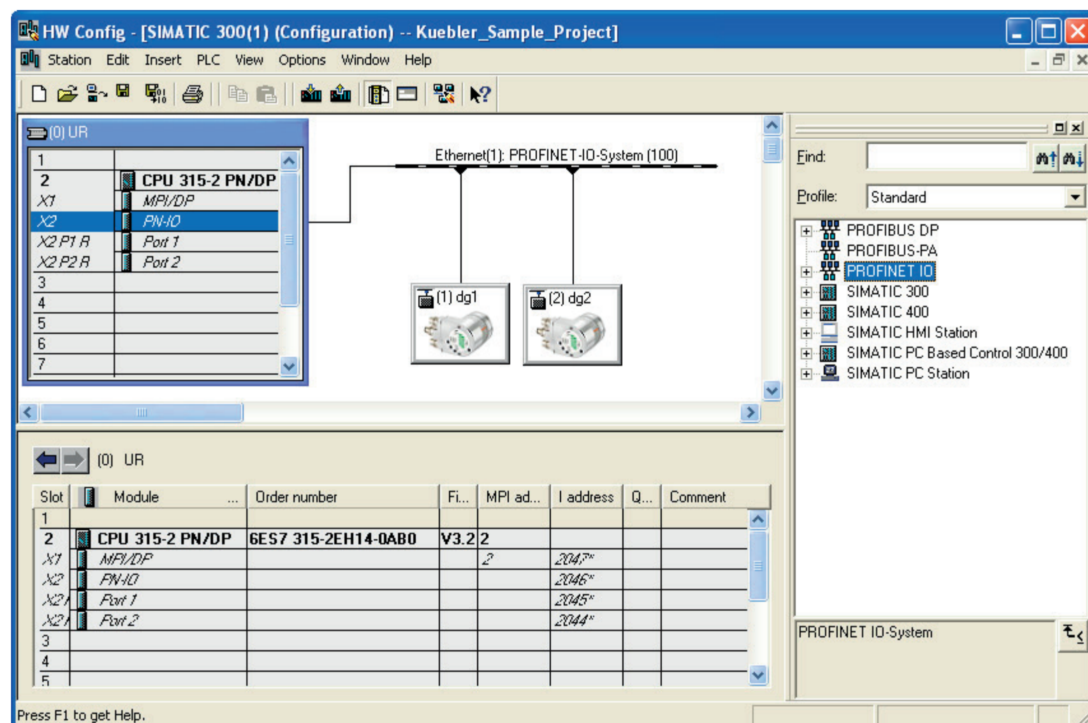
☑ Assign IP address via IO controller

Comment:

OK Cancel Help

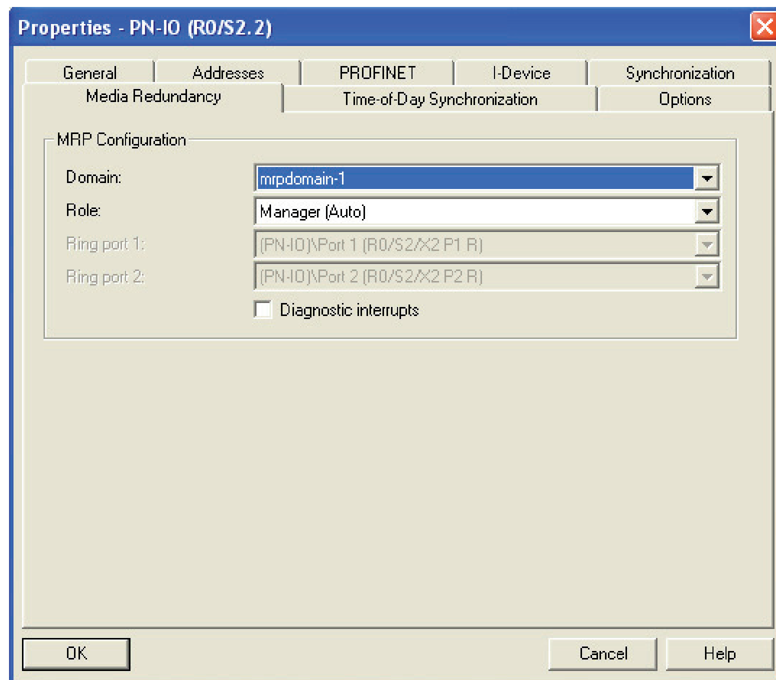
Picture 22

Configuration of CPU315 2PN/DP for MRP operation



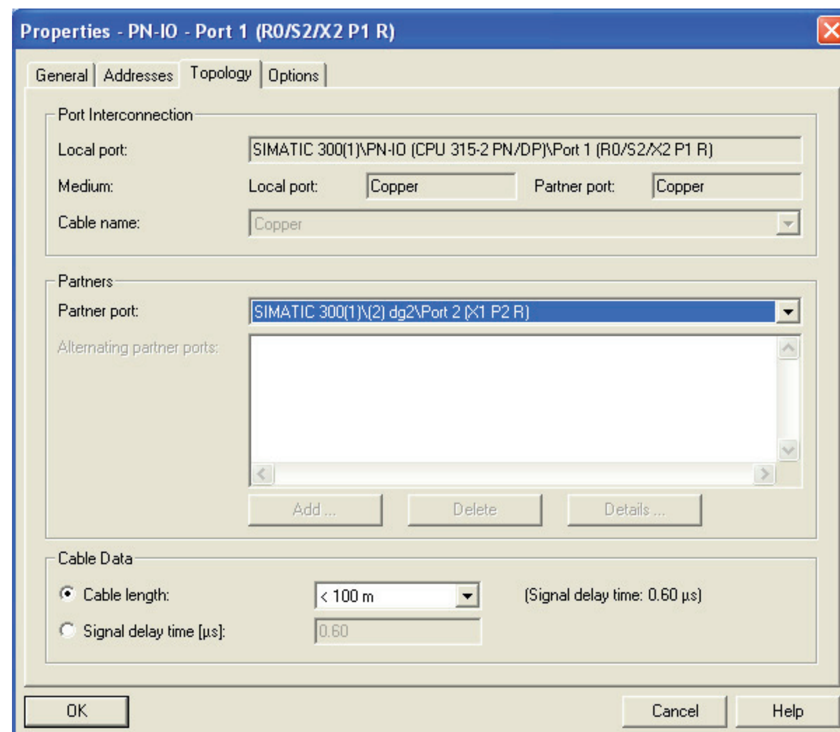
Picture 23

As shown in picture 23, double-click on line “PN-IO”. In the dialogue window that opens, set all parameters according to picture 24. In particular, CPU315 must be configured as an MRP manager.



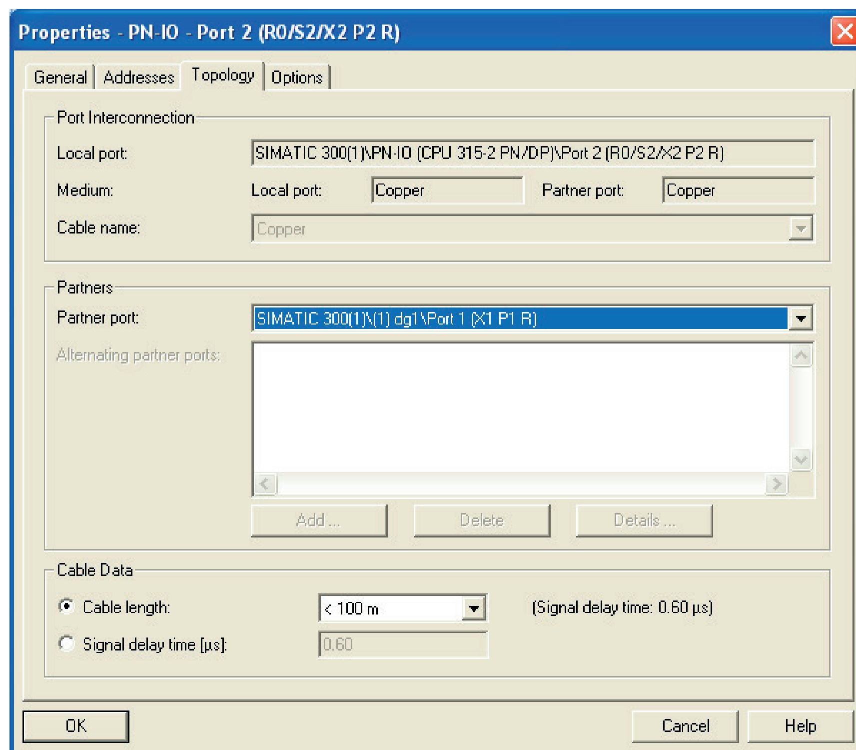
Picture 24

Double-clicking on line “Port1” opens the dialogue window represented in picture 25, whose parameters must be set as follows. Port 1 of CPU315 is then connected to port 2 of DG2.



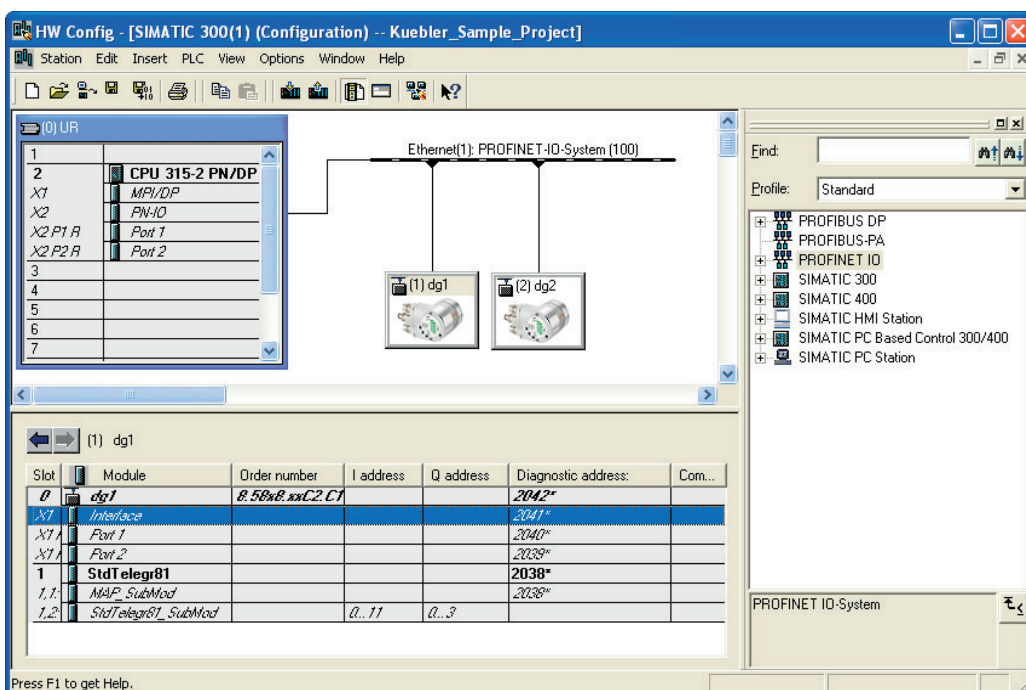
Picture 25

The following dialogue window applies to port 2. Port 2 of CPU315 is then connected to port 1 of DG1.



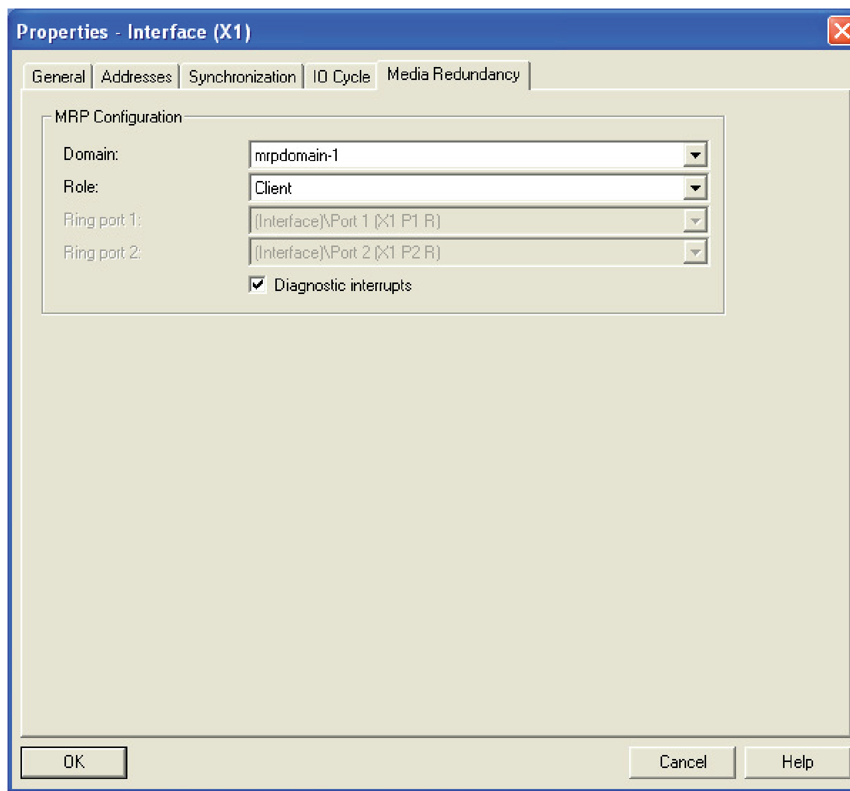
Picture 26

Configuration of both encoders for MRP operation
The same as above (CPU configuration) applies to DG1, starting from picture 27.



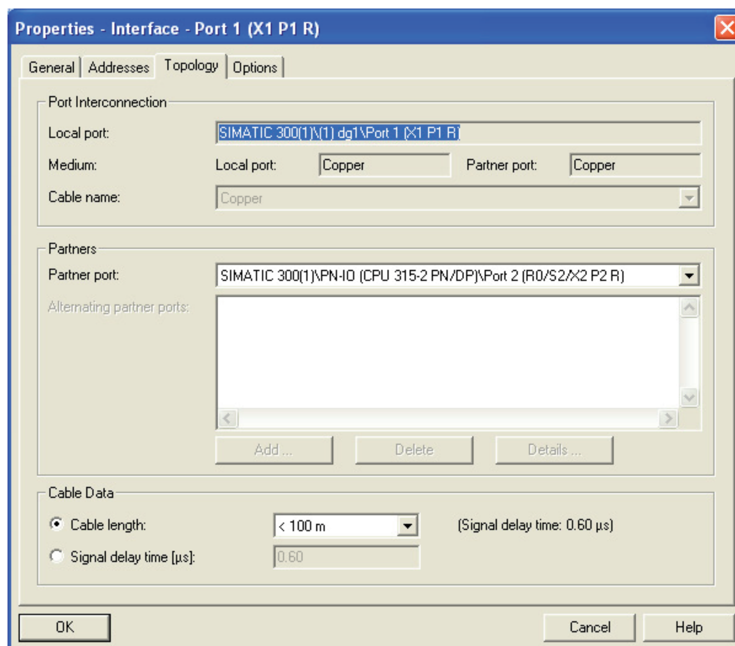
Picture 27

Double-clicking on line “Interface” of DG1 opens the dialogue window represented in picture 28. DG1 is an MRP client and must be parameterised as such.

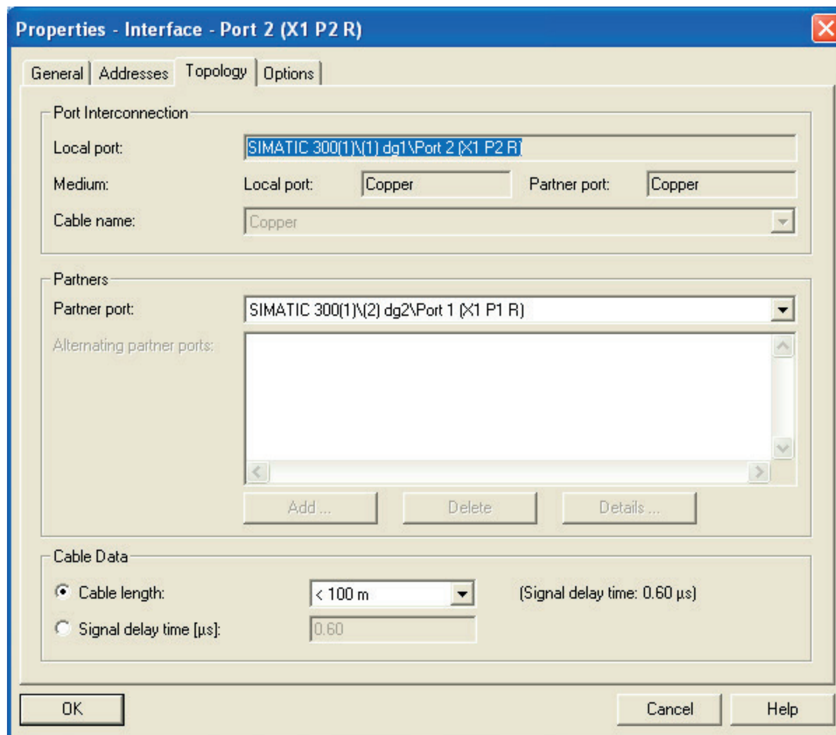


Picture 28

The settings of pictures 29 and 30 apply to both ports 1 and 2 of DG1.



Picture 29



Picture 30

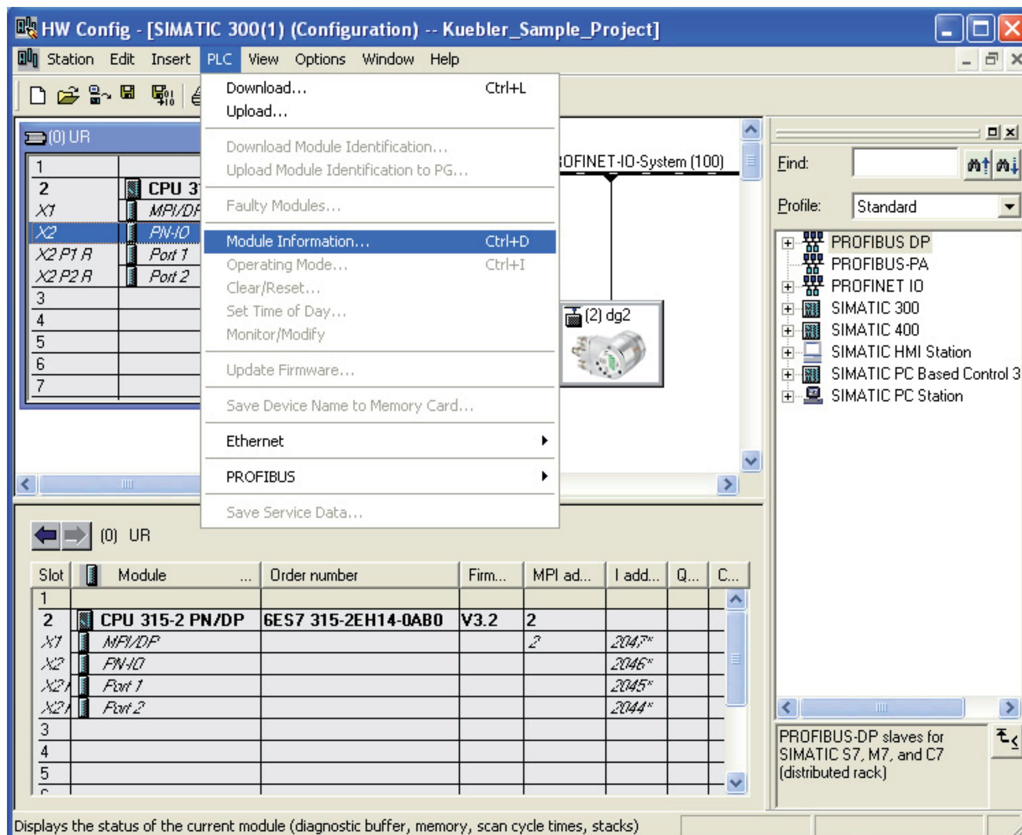
As DG1, DG2 must be configured as an MRP client.

Upon a download via port 1 of CPU315, port 1 is connected to port 2 of DG2. CPU315 must then be restarted by means of the reset switch.

Upon the interruption of the ring, e.g. at port 1 of the CPU, the CPU displays an error with the corresponding LED, but it does not switch to Stop mode. Once the ring is closed again, the error message disappears.

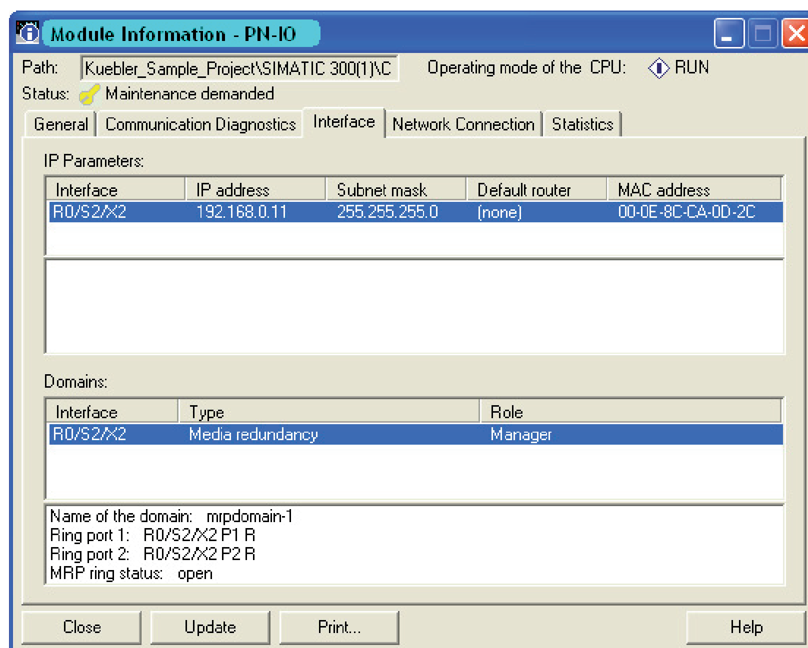
In case on the interruption of the ring, the STEP7 PC can be connected, and the error diagnosis can be performed as follows.

As shown in picture 31, select line “PN-IO” of CPU315 and, in menu “PLC”, select option “Module Information…”.

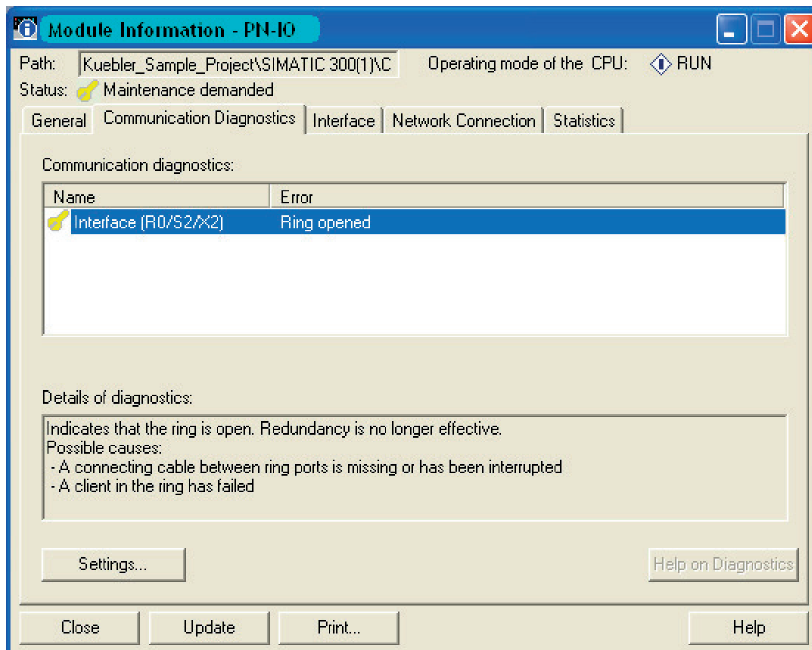


Picture 31

A dialogue window then opens, showing both the condition of the component and the status of the communication. This is shown on both pictures 32 and 33.

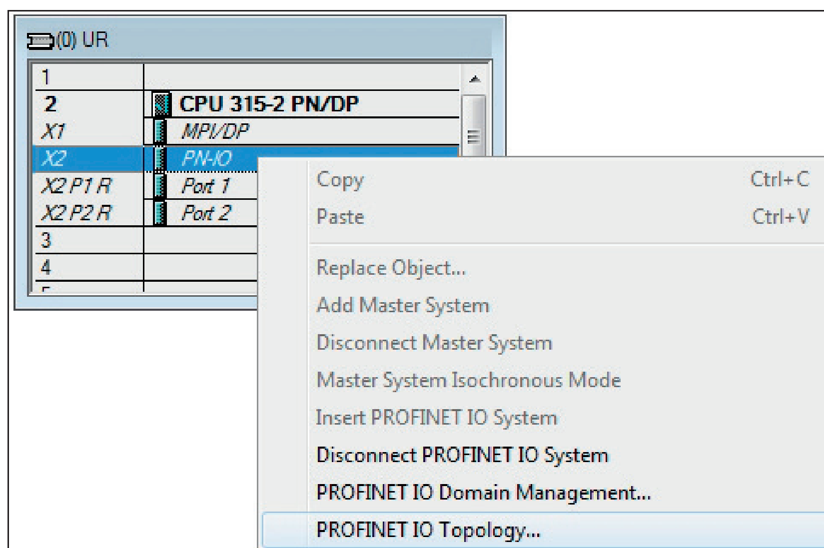


Picture 32

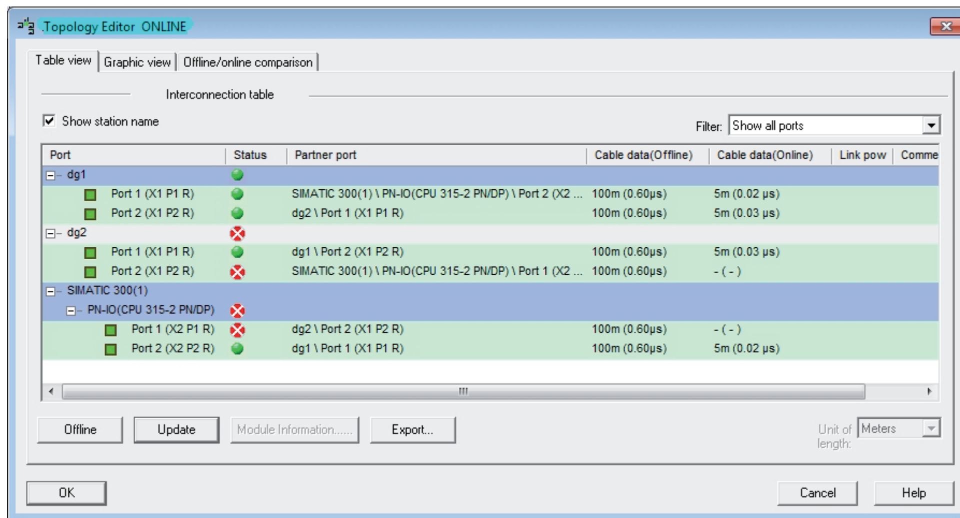


Picture 33

As an alternative, a general view of the dialogue window can be obtained selecting menu “PROFINET IO Topology...”. This window, represented in picture 35, shows that port 1 of the CPU and port 2 of DG2 have an error.



Picture 34



Picture 35

Appendix A: Reading / writing the Prm 65000 preset value

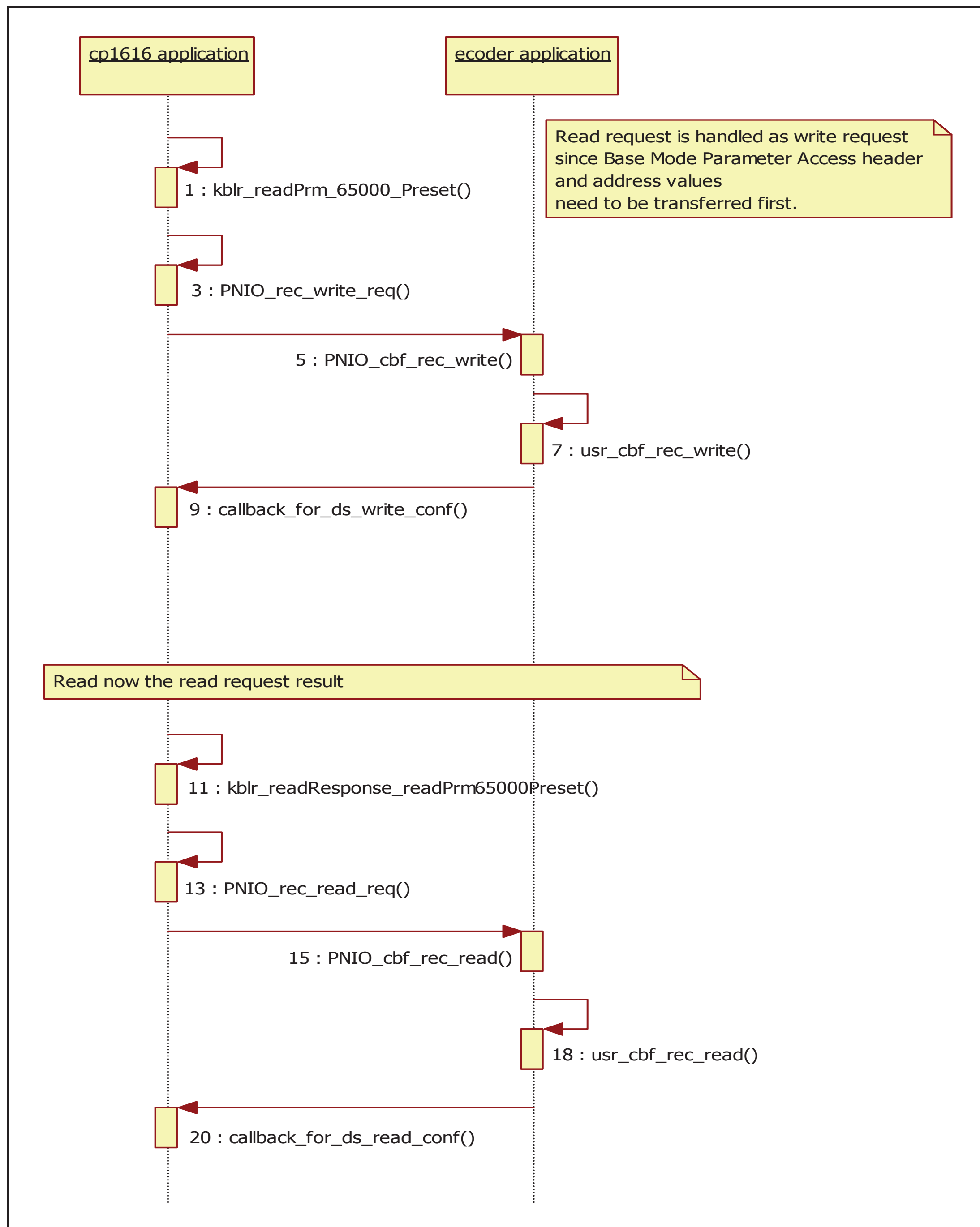
The mechanism for writing / reading of Preset Parameter 65000 is known as "Base Mode Parameter Access" (BMPA) and is described in detail on page 59 and following pages of the PROFIdrive profile specification.

Picture 36 depicts a message sequence chart of parameter 65000 reading. All function names starting with "kblr_" are Kübler specific and may be replaced by appropriate user function names. All other functions represent functionality out of the PROF-INET – API for the CP1616 controller as given by SIEMENS and should not be changed.

The read request for Prm 65000 within BMPA is in fact a write request followed by a read request.

Function kblr_readPrm_65000_Preset populates all structs according to BMPA and hands them over to PNIO_rec_write_req.

The corresponding code snippets are given on the following pages. A similar situation applies for a change request of Prm 65000 within BMPA, which is a combination of a write and read request to the BMPA index.


cp1616 application
ecoder application
Read request is handled as write request
since Base Mode Parameter Access header
and address values
need to be transferred first.
1 : kblr_readPrm_65000_Preset()
3 : PNIO_rec_write_req()
5 : PNIO_cbf_rec_write()
7 : usr_cbf_rec_write()
9 : callback_for_ds_write_conf()
Read now the read request result
11 : kblr_readResponse_readPrm65000Preset()
13 : PNIO_rec_read_req()
15 : PNIO_cbf_rec_read()
18 : usr_cbf_rec_read()
20 : callback_for_ds_read_conf()

Picture 36

The structs are given as follows:

```
typedef struct
{
	PNIO_UINT8	RequestID;
	PNIO_UINT8	RequestRef;
	PNIO_UINT8	NoOfPrms;
	PNIO_UINT8	DO_ID;

} sBMPReqHeader;

typedef struct
{
	PNIO_UINT8	NoOfElements;
	PNIO_UINT8	Attribute;
	PNIO_UINT16	PNU;
	PNIO_UINT16	SubIdx;
} sBMPPrmAddress;

typedef struct
{
	PNIO_UINT8	NoOfValues;
	PNIO_UINT8	Format;
} sBMPPrmValue;

typedef struct
{
	PNIO_UINT8	ResponseID;
	PNIO_UINT8	RequestRefMir;
	PNIO_UINT8	NoOfPrms;
	PNIO_UINT8	DO_IDMir;
}sBMPRespHeader;

typedef struct
{
	sBMPRespHeader	BMPRespHeader;
	sBMPPrmValue	BMPPrmValue;
	PNIO_UINT8	valArray [sizeof (PNIO_UINT32)];
} sBMPResponseBuf;

typedef enum
{
	KBLR_REQUEST_PARAMETER	=	1,
	KBLR_CHANGE_PARAMETER	=	2
} KBLR_RequestID;

typedef enum
{
	KBLR_VALUE	=	0x10,
	KBLR_DESCRIPTION	=	0x20,
	KBLR_TEXT	=	0x30
} KBLR_Attribute;
```

```
typedef enum
{
      KBLR_ZERO    =  0x40,
      KBLR_BYTE    =  0x41,
      KBLR_WORD    =  0x42,
      KBLR_DWORD   =  0x43,
      KBLR_ERROR   =  0x44
} KBLR_Format;

typedef enum
{
      KBLR_REQUEST_PARAMETER_P    =  0x01,
      KBLR_CHANGE_PARAMETER_P     =  0x02,
      KBLR_REQUEST_PARAMETER_M    =  0x81,
      KBLR_CHANGE_PARAMETER_M     =  0x82
} KBLR_ResponseID;

#define KBLR_BASEMODEPRMACCESS_INDEX 0xB02E

/****************************************************************** */
/* This function acyclically reads parameter                        */
/* Prm_65000_Preset from encoder                                    */
/****************************************************************** */

void kblr_readPrm_65000_Preset (void)
{
      sBMPReqHeader         BMPReqHeader;
      sBMPPrmAddress        BMPPrmAddress;
      PNIO_UINT8*           pMem8 = NULL;

      // The logical address of the MAP/PAP submodule is 0
      // in this example
      PNIO_ADDR             SubModAddress = { PNIO_ADDR_LOG, PNIO_IO_OUT, 0 };
      PNIO_UINT32           dwErrorCode;
      PNIO_UINT32           RecordIndex = KBLR_BASEMODEPRMACCESS_INDEX;
      PNIO_REF              ReqRef = 1;

      // Now we fill the BMPA request for single value struct
      BMPReqHeader.RequestID    =   KBLR_REQUEST_PARAMETER;
      BMPReqHeader.RequestRef   =   0xAB; // To be mirrored by encoder
      BMPReqHeader.NoOfPrms     =   0x01
      BMPReqHeader.DO_ID        =   0xCD; // To be mirrored by encoder

      BMPPrmAddress.NoOfElements    =   0x00;
      BMPPrmAddress.Attribute       =   KBLR_VALUE;
      BMPPrmAddress.PNU             =   OsHtons(65000); // BIG ENDIAN !

      pMem8 = (PNIO_UINT8*)malloc(sizeof (sBMPReqHeader)+
                                  sizeof (sBMPPrmAddress));

      memcpy(pMem8,(PNIO_UINT8*)&BMPReqHeader, sizeof (sBMPReqHeader));
      memcpy(pMem8 + sizeof (sBMPReqHeader),(PNIO_UINT8*)&BMPPrmAddress,
                                  sizeof (sBMPPrmAddress));
```

```
dwErrorCode = PNIO_rec_write_req (
                    g_dwHandle,        // handle
                    &SubModAddress, // Address of the submodule
                    RecordIndex,
                    ReqRef,
                    sizeof (sBMPReqHeader) + sizeof (sBMPPrmAddress),
                    (PNIO_UINT8*)pMem8);
    free(pMem8);
}
```

On execution of callback_for_ds_write_conf we might investigate the returned error code and decide if the requested value can now be "picked up" by doing a BMPA read request.

The following function code shows how to do this:

```
/**********************************************************************/
/* This function should immediately be called after
kblr_readPrm_65000_Preset in order to read out the result of
'parameter request' request for 65000_Preset parameter. We
/* simply do a read to index 0xB02E which is the BMPA index
/**********************************************************************/

void kblr_readResponse_readPrm65000Preset (void)
{
    // The logical address of the MAP/PAP submodule
    // is 0 in this example.
    PNIO_ADDR       SubModAddress = { PNIO_ADDR_LOG, PNIO_IO_OUT, 0 };
    PNIO_UINT32     dwErrorCode;
    PNIO_UINT32     RecordIndex = KBLR_BASEMODEPRMACCESS_INDEX;
    PNIO_REF        ReqRef = 1;
    dwErrorCode = PNIO_rec_read_req(
                    g_dwHandle,         // handle
                    &SubModAddress,     // Address of the submodule
                    RecordIndex,
                    ReqRef,
                    sizeof (sBMPResponseBuf));
}
```

Since callback_for_ds_read_conf provides a pointer to parameter data and error code the cycle of requesting and responding is closed. The following text represents a code snippet out of callback_for_ds_read_conf.

```
Case KBLR_BASEMODEPRMACCESS_INDEX:
{
        printf („\r\ncallback_for_ds_read_conf: Receiving BMP Access data:
                0x%04x\n“, pCbfPrm->RecWriteConf.RecordIndex);

        if (pCbfPrm->RecReadConf.Err.ErrCode == 0xDE)
        {
                printf („\r\nNo BMPA mode response available yet!\r\n“);
        }
        if ( (pCbfPrm->RecReadConf.Err.ErrCode == 0) &&
                (pCbfPrm->RecReadConf.Length > 0))
        {
                for (i=0; i<pCbfPrm->RecReadConf.Length; i++)
        {
                // We simply print the received bytes
                printf („pBuf[%02d]=%02x\t“,i,
                                        *(pCbfPrm->RecReadConf.pBuffer+i));
                }
                printf(„\r\n“);
        }
}
break;
```

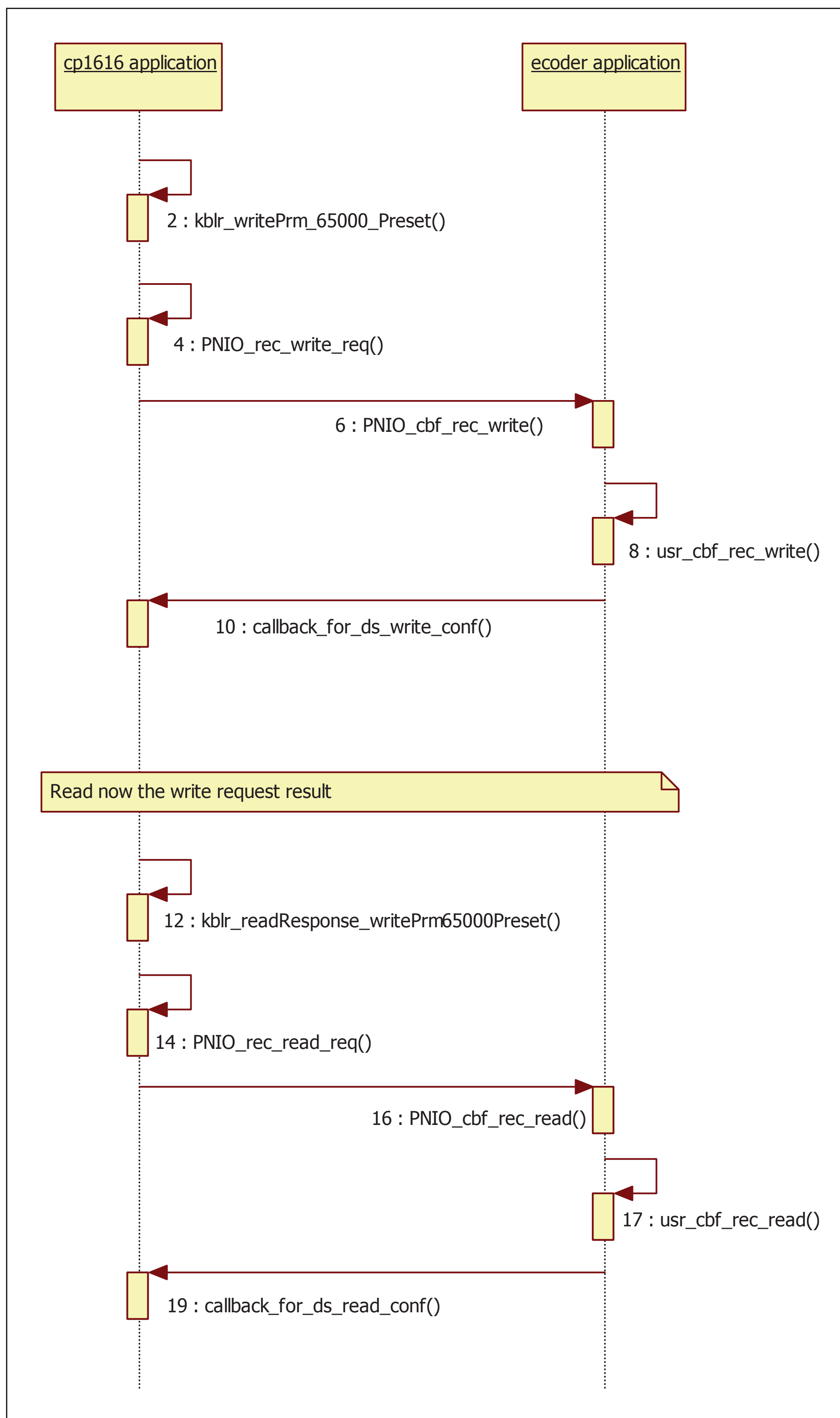
Important!



The callback function callback_for_ds_read_conf should return as soon as possible. Time consuming operations like printf should actually be moved to threads executing in a lower priority context.

Picture 37 depicts a message sequence chart which describes the scenario of writing Prm 65000.

English



Picture 37

```
/***********************************************************************/
/* This function acyclically writes parameter Prm_65000_Preset to      */
   encoder                                                              */
/***********************************************************************/

void kblr_writePrm_65000_Preset (int32_t i32Preset)
{
      sBMPReqHeader        BMPReqHeader;
      sBMPPrmAddress       BMPPrmAddress;
      sBMPPrmValue         BMPPrmValue;
      PNIO_UINT8*          pMem8 = 0;

      // The logical address of the MAP/PAP submodule
      // is 0 in this example
      PNIO_ADDR            SubModAddress = { PNIO_ADDR_LOG, PNIO_IO_OUT, 0 };
      PNIO_UINT32          dwErrorCode;
      PNIO_UINT32          RecordIndex = KBLR_BASEMODEPRMACCESS_INDEX;
      PNIO_REF             ReqRef = 1;

      // Now we fill the BPMA request for single value struct
      BMPReqHeader.RequestID          = KBLR_CHANGE_PARAMETER;
      BMPReqHeader.RequestRef         = 0x12; // Should be mirrored by
                                              // encoder
      BMPReqHeader.NoOfPrms           = 0x01;
      BMPReqHeader.DO_ID              = 0x34;  // Should be mirrored by
                                              // encoder
      BMPPrmAddress.NoOfElements      = 0x01;
      BMPReqHeader.DO_ID              = 0x34; // Should be mirrored by
                                              // encoder
      BMPPrmAddress.NoOfElements      = 0x00;
      BMPPrmAddress.Attribute         = KBLR_VALUE;
      MPPrmAddress.PNU                = OsHtons(65000); // BIG ENDIAN !

      BMPPrmValue.NoOfValues          = 1;
      BMPPrmValue.Format              = KBLR_DWORD;

      i32Preset = OsHtonl(i32Preset); // BIG ENDIAN !!!

      pMem8 = (PNIO_UINT8*)malloc (sizeof (sBMPReqHeader)+
               sizeof (sBMPPrmAddress) + sizeof (sBMPPrmValue)+
               sizeof (int32_t));

      memcpy(pMem8, (PNIO_UINT8*)&BMPReqHeader,
               sizeof (sBMPReqHeader));
      memcpy(pMem8+sizeof(sBMPReqHeader),
               (PNIO_UINT8*)&BMPPrmAddress, sizeof (sBMPPrmAddress));

      memcpy(pMem8 + sizeof (sBMPReqHeader) + sizeof (sBMPPrmAddress),
               (PNIO_UINT8*)&BMPPrmValue, sizeof (sBMPPrmValue));

      memcpy(pMem8 + sizeof (sBMPReqHeader) + sizeof (sBMPPrmAddress)+
               sizeof (sBMPPrmValue), &i32Preset, sizeof (int32_t));
```

```
        dwErrorCode = PNIO_rec_write_req(
                g_dwHandle,                 // handle
                &SubModAddress, // Address of the submodule
                RecordIndex,
                ReqRef,
                sizeof (sBMPReqHeader) + sizeof (sBMPPrmAddress) +
                sizeof (sBMPPrmValue) + sizeof (int32_t),
                                            (PNIO_UINT8*)pMem8);
                free(pMem8);
        }
```

In callback_for_ds_write_conf it is possible to evaluate a possible returned error code and to decide whether to read the result of the change request operation.

```
/**********************************************************************/
/* This function should immediately be called after                  */
/* kblr_writePrm_65000_Preset in order to read out the result of a   */
/* parameter change request for 65000_Preset parameter.              */
/* We simply do a read to index 0xB02E which is the BMPA index       */
/**********************************************************************/

void kblr_readResponse_writePrm65000Preset (void)
{
        // The logical address of the MAP/PAP submodule
        // is 0 in this example
        PNIO_ADDR                   SubModAddress = { PNIO_ADDR_LOG, PNIO_IO_OUT, 0 };
        PNIO_UINT32 dwErrorCode;
        PNIO_UINT32 RecordIndex = KBLR_BASEMODEPRMACCESS_INDEX;
        PNIO_REF                    ReqRef = 1;
        dwErrorCode = PNIO_rec_read_req(
                g_dwHandle,             // handle
                &SubModAddress,          // Address of the submodule
                RecordIndex,
                ReqRef,
                sizeof (sBMPResponseBuf));
}
```

The BMPA index code snippet out of callback_for_ds_read_conf applies in this case too. However in the first case we read the preset value itself whereas here, in the second case, we simply read the result of the change request operation.

References

1. PROFINET Cabling and Interconnection Technology Guideline Version 2.00 March 2007
Order No: 2.252

2. Profile Encoder, Technical Specification for Profibus and PROFINET related to PROFIdrive Version 4.1 December 2008
Order No: 3.162

3. Profile Drive Technology PROFIdrive Technical Specification for PROFIBUS and PROFINET Version 4.1 May 2006.
Order No: 3.172

www.kuebler.com



Kübler Group
Fritz Kübler GmbH
Schubertstrasse 47
D-78054 Villingen-Schwenningen
Germany
Phone: +49 7720 3903-0
Fax: +49 7720 21564
info@kuebler.com
www.kuebler.com

pulses for automation

R.60713.0002_0

Kübler



Notice

Sendix 5858/5878 absolu monotour
Sendix 5868/5888 absolu multitours

Pour Réf. de commande 8.58X8.XXCX.C2XX
à partir de la version de firmware 2.0

Pour Réf. de commande 8.58X8.XXCX.C1XX
à partir de la version de firmware 1.37

■■■ *l'impulsion pour l'automatisme*



Réserve de modifications
Dans le cadre de nos efforts d'amélioration permanente de nos produits, nous nous réservons le droit d'apporter à tout moment des modifications techniques aux informations techniques contenues dans le présent document.

Aucune garantie
Fritz Kübler GmbH ne donne aucune garantie, implicite ou explicite, en rapport avec l'ensemble de la présente notice (original en langue allemande et traduction en langue française), et décline toute responsabilité en cas de dommages directs ou indirects. Les propriétés du produit et les caractéristiques techniques indiquées ne constituent en aucun cas une déclaration de garantie.

Informations sur le document
Indice de modification 07/2013
Notice originale. Original en langue allemande.

Groupe Kübler
Fritz Kübler GmbH.
Schubertstrasse 47
78054 Villingen-Schwenningen
Allemagne
Tél. : +49 7720 3903-0
Télécopie : +49 7720 21564
info@kuebler.com
www.kuebler.com

Sommaire

Version de firmware et fichier GSDML

Versions les plus récentes du firmware du codeur et du fichier GSDML lors de l'édition du présent document :
Version de firmware V2.00 GSDML-V2.2-KUEBLER-0198-Sendix58xxPNIO-20130116-131800.xml

Caractéristiques techniques des codeurs

Caractéristiques mécaniques

Résistance aux chocs selon EN 60068-2-27	2500 m/s², 6ms pour les versions monotour 2000 m/s², 6ms pour les versions multitours
Résistance aux vibrations selon EN 60068-2-6	100m/s², 10.....2000 Hz

Température de fonctionnement
-40…+85°C

Alimentation électrique
10…30 VDC

200 mA sous 10 VDC
80 mA sous 24 VDC
60 mA sous 30 VDC

Caractéristiques hardware
ASIC PROFINET IO : ERTEC 200
Autonégociation
Polarité automatique
Croisement automatique
LED de signalisation de fonctionnalité et de diagnostic

Normes et protocoles supportés
RT_CLASS_1
RT_CLASS_2
RT_CLASS_3 (IRT)
DCP
RTA
LLDP
SNMP
MIB-II et LLDP-MIB
PTCP
MRP



Profil codeur implémenté
Profil codeur version 4.1

Fonctionnalité d'identification et de maintenance
Version 1.2
Blocs I&M supportés 0, 1, 2, 3, 4

Conformité à
EN 61000-4-2 :2001
EN 61000-4-3 :2006
EN 61000-4-4 :2005
EN 61000-4-5 :2007
EN 61000-4-6 :2008
EN 61000-4-7 :2004

EN 61000-6-4 :2007
EN 61000-6-2 :2006

Installation

L'installation d'un codeur comporte cinq étapes :

1. Installation du câblage des données
2. Installation du câblage d'alimentation
3. Configuration à l'aide de SIMATIC NCM PC ou de STEP 7
4. Installation d'un contrôleur PROFINET
5. Lancement de l'application du contrôleur avec les codeurs

Installation du câblage des données

Le codeur est muni de trois connecteurs, dont deux sont des ports Ethernet. Ils sont désignés dans cette documentation respectivement comme port 1 et port 2; ils sont repérés par des flèches sur l'autocollant, comme représenté sur la figure 1 ci-dessous. Le connecteur central est le connecteur d'alimentation ; il sera décrit dans le chapitre suivant.



Illustration 1

Les connecteurs des ports 1 et 2 sont des connecteurs femelle M12 à 4 broches, avec un codage D. La figure 2 et le tableau ci-dessous indiquent l'affectation des broches.

Affectation des signaux pour un connecteur femelle M12 avec codage D

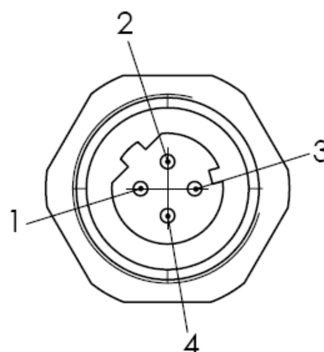


Illustration 2 : M12 femelle, 4 broches, codage D

Signal sur conn. femelle M12 4 broches avec codage D	Fonction	Couleur de fil	N° de broche
TD+	Emission de données +	Jaune	1
TD-	Emission de données -	Orange	3
RD+	Réception de données +	Blanc	2
RD-	Réception de données -	Bleu	4

Affectation des signaux et des broches pour un câble RJ45 sur M12
M12 sur RJ45 droit

Signal	M12	RJ45
TD+	1	1
TD-	3	2
RD+	2	3
RD-	4	6

M12 sur RJ45 croisé

Signal	M12	RJ45
TD+	1	3
TD-	3	6
RD+	2	1
RD-	4	2

Câble préconisé pour le câblage PROFINET :
Câble souple industriel Ethernet FC TP Siemens,
GP 2x2 (PROFINET Type B), installation par paires torsadées
N° de commande : 6XV1870-2B

Connecteur RJ45 préconisé :
Siemens IE FC RJ45 N° de commande : 6GK1901-1BB10-2AA0

Important !



PROFINET est basé sur la technologie Fast Ethernet. De ce fait, la longueur maximale admissible des segments est de 100 m. Dans le cas de distances supérieures à 100 m, il faut insérer des commutateurs. Ne pas utiliser de hubs ! Les commutateurs doivent être certifiés selon les spécifications PROFINET. Si le codeur est configuré pour un fonctionnement MRP (Media Redundancy Protocol), les commutateurs insérés doivent être commandés et, pour le mode IRT60, ils doivent tous être compatibles IRT ! Exemple d'un commutateur à la fois commandé et compatible IRT : SCALANCE 200 IRT.

Installation du câblage d'alimentation

La figure 3 et le tableau ci-dessous décrivent l'affectation des broches du connecteur mâle 4 broches avec codage A d'alimentation du codeur.

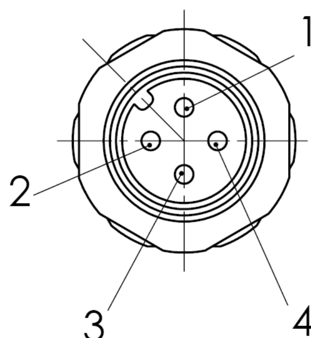


Illustration 3 : M12 mâle, 4 broches, codage A

Signal sur conn. mâle M12 à 4 broches avec codage A	Fonction	N° de broche
PWR	10 – 30 V DC	1
		2
GND	0V	3
		4

Pour davantage de détails sur le câblage dans des environnements PROFINET, voir la spécification PN-Cabling-Guide_2252_V200_May07.pdf.

Cette spécification peut être téléchargée de la zone en accès libre à l'adresse http://www.profibus.com/downloads/

LED de diagnostic

Les codeurs sont munis de quatre LED de diagnostic repérées comme indiqué sur la figure ci-dessous.



Illustration 4

Marquage	Couleur	Description de la fonctionnalité
LINK 1	Jaune et verte	LINK 1 est une LED bicolore pour le port 1 indiquant l'activation de la liaison (vert) et le transfert de données (jaune)
LINK 2	Jaune et verte	LINK 2 est une LED bicolore pour le port 2 indiquant l'activation de la liaison (vert) et le transfert de données (jaune)

Le tableau ci-dessous décrit toutes les situations de fonctionnement signalées par des combinaisons des LED ERROR et PWR.

LED ERROR (rouge)	LED PWR (verte)	Signification	Cause possible
Eteinte	Allumée	Fonctionnement normal. Echange de données OK.	
Clignotante	Allumée	L'échange de données sur le bus est possible, mais le codeur n'est pas passé dans le mode d'échange de données process. Le codeur indique un numéro d'erreur dans G1_XIST2. En alternative, le code d'erreur peut être lu dans le paramètre 65001.	Voir les codes d'erreur clignotants dans le chapitre suivant.
Allumée	Allumée	L'échange de données sur le bus est possible, mais aucun échange de données n'a lieu sur le bus.	Maître indisponible ou bus débranché.
Eteinte	Eteinte	Pas d'alimentation électrique	

Codes d'erreur clignotants

Code de clignotement	Cause
Toutes les 2 secondes (0,5 Hz) LED Error allumée 1 seconde, éteinte 1 seconde, puis répétition	- L'esclave n'a pas encore été configuré par le contrôleur. Le codeur n'a pas encore reçu les données des paramètres utilisateur sous la forme du jeu de données index 0xBF00 - Erreur de configuration - Affectation d'une adresse de station erronée (mais dans la plage d'adresses permise) - Configuration réelle de l'esclave différente de sa configuration nominale.
5 fois par seconde LED Error allumée 0,1 seconde, éteinte 0,1 seconde, puis répétition	Communication bus OK, mais communication interrompue entre le codeur et le capteur des données de position.
Une fois par seconde (1Hz) LED Error allumée 0,5 seconde, éteinte 0,5 seconde, puis répétition	Erreur mémoire

Exemple de configuration d'un projet à l'aide de STEP 7

Important !



Pour la configuration du projet, utiliser impérativement STEP7 version 5.5. Dans le cas contraire, des erreurs surviendront lors de l'installation du fichier GSDML du codeur et de tous les autres fichiers GSDML utilisant le même schéma XML. En outre, des erreurs surviendront lors de la configuration et du paramétrage MRP.

Un exemple de projet STEP 7 nommé "Kuebler_Sample_Project" peut être téléchargé depuis le serveur Internet de Kübler. Ce projet sera décrit dans les pages suivantes. Il faut également installer le fichier GSDML du codeur PROFINET MRP avant de démarrer le codeur sous STEP7.

Les deux illustrations suivantes représentent le projet lui-même et la configuration hardware avec deux codeurs qui utilisent tous deux le même DAP, mais avec des modules différents.

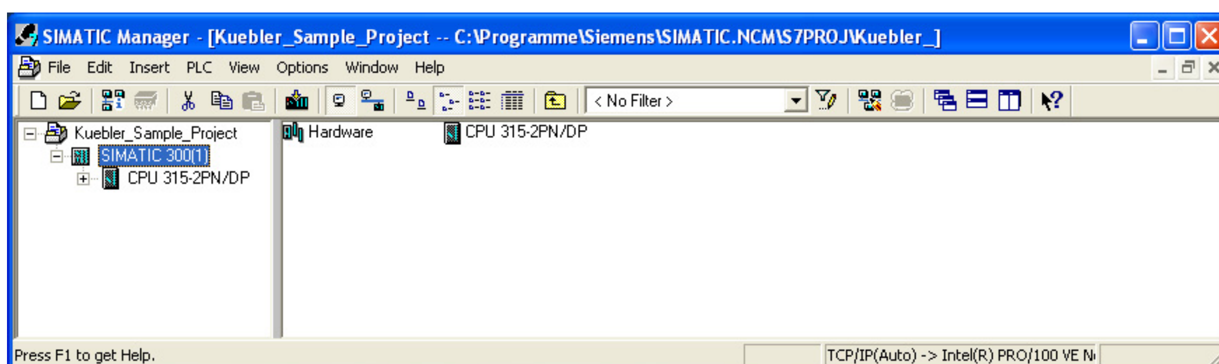


Illustration 5

Le premier codeur dg1 utilise le module avec le télégramme standard 81 représenté dans l'illustration 6. Son adresse pour la lecture du télégramme standard 81 est 0. L'adresse pour l'écriture du télégramme standard 81 qui permet, entre autres, de déclencher le prépositionnement, est également 0.

Le second codeur dg2 utilise le module qui inclut à la fois le télégramme standard 81 et la vitesse. Les deux adresses pour la lecture et l'écriture du télégramme standard 81 sont 12 et 4, comme représenté dans l'illustration 7



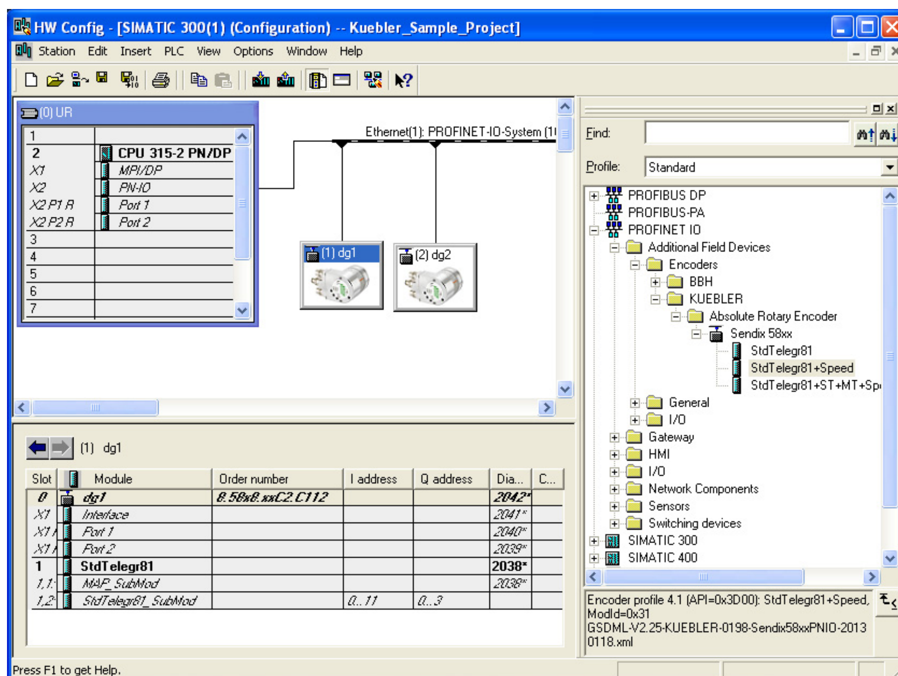


Illustration 6

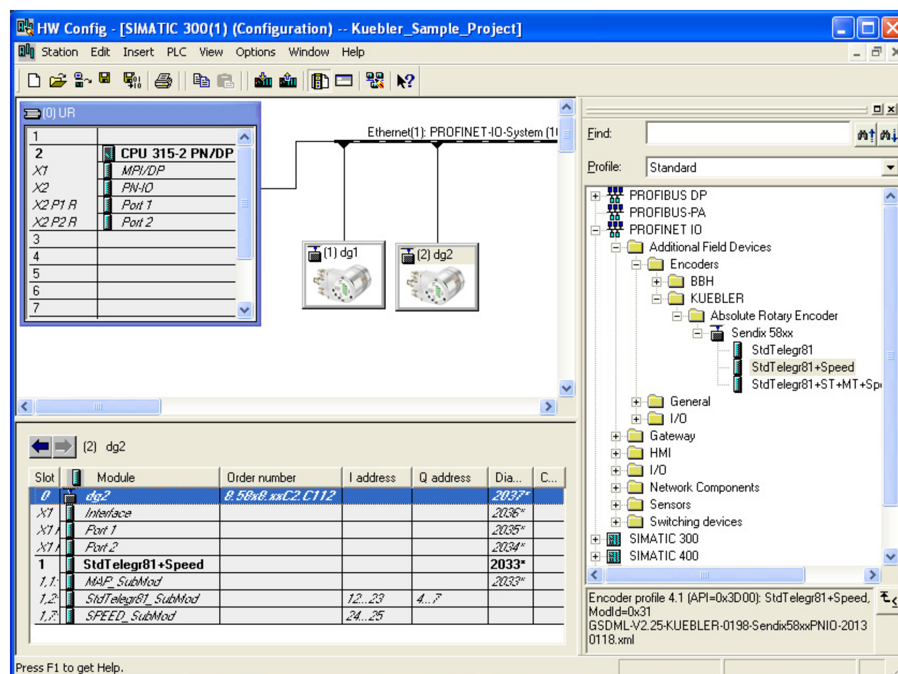


Illustration 7

Configuration des paramètres utilisateur du codeur

Les données des paramètres utilisateur sont définies à la page 52 de la spécification du profil codeur; elles peuvent être envoyées au codeur sous la forme d'un objet Enregistrement de données au cours de sa phase de démarrage. La copie d'écran de dialogue ci-dessous indique les différentes données de cet enregistrement.

Pour lancer le dialogue de configuration des paramètres, double-cliquer sur la ligne "MAP_SubMod" dans la configuration hardware du codeur concerné.

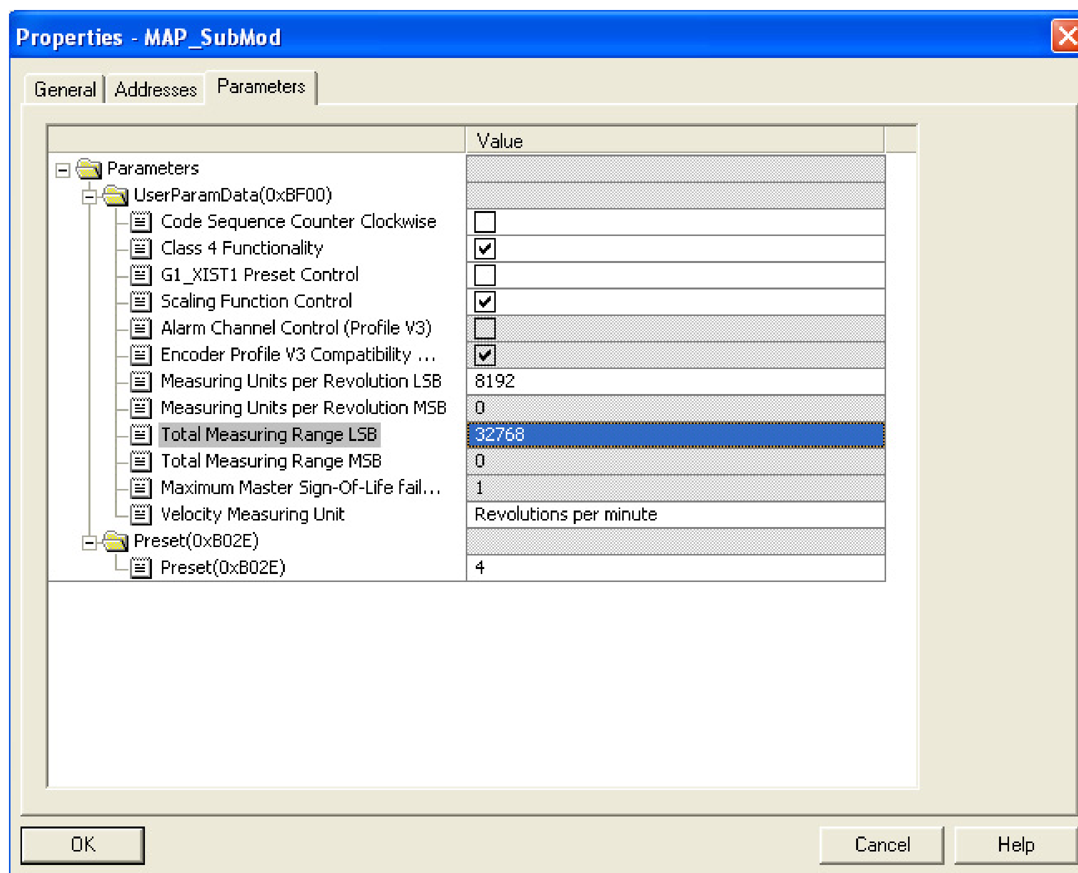


Illustration 8

Les valeurs des paramètres sont transmises de l'API au codeur lors de la phase de préparation du modèle de machine d'état de base du codeur.

Pour lancer le dialogue, double-cliquer sur la ligne intitulée "MAP-SubMod", qui représente la zone 1 sous-zone 1 du dialogue de configuration du codeur. Comme le type de codeur Sendix est un codeur avec une résolution monotour de 16 bits et une résolution multitours de 12 bits, les quatre octets de poids le plus fort correspondants de Measuring Units per Revolution (MUR - unités de mesure par tour) et de Total Measuring Range (TMR - plage de mesure totale) ne peuvent pas être modifiés. Ils s'affichent par conséquent sur un fond gris et ont une valeur nulle par défaut.

Les valeurs de l'illustration 10 sont vérifiées automatiquement lorsque l'utilisateur clique sur le bouton OK. Si nécessaire, STEP7 affiche sa propre plage de valeurs et demande à l'utilisateur de corriger ses valeurs.

Pour MUR et TMR, le plage de valeurs dépend des points suivants :

- La valeur de MUR (measuring units per revolution –unités de mesure par tour) ne sera acceptée que si elle répond aux critères suivants :
  $0 < MUR \leq g_ST$
  où g_ST correspond à la résolution monotour physique (65536 pour 16 bits).

- Pour un codeur **sans** unité multitours, la valeur de TMR (total measuring range – plage de mesure totale) ne sera acceptée que si elle répond aux critères suivants :
  $0 < TMR \leq g_ST$
  où g_ST correspond à la résolution monotour physique, qui est de 65536 (16 bits).

- Les codeurs **avec** unité multitours doivent répondre aux critères suivants :
  $0 < TMR \leq MUR * g_MT$
  où MUR correspond au nombre d'unités de mesure par tour et g_MT à la résolution multitours physique (4096 pour 12 bits) pour le type Sendix décrit dans ce document.

Attention !



Les valeurs de résolution monotour et multitours doivent être des valeurs correspondant à la formule 2^X (2 à la puissance X). Dans ce cas, il n'y aura pas de reste inférieur à la valeur monotour lorsque l'unité multitours atteint sa valeur maximale, indiquée par la résolution physique multitours maximale.

Les critères suivant s'appliquent à la valeur de prépositionnement :
- $0 \leq$ Prepositionnement $\leq$ valeur TMR ET
- Prépositionnement $\leq$ 0x7FFFFFFF (MaxINT)

Lecture des valeurs de position du coedur

La fonctionnalité du bloc OB1 a été implémentée afin de représenter de manière très simple le principe de la lecture de la valeur de position. Dans l'illustration 9 ci-dessous, les deux réseaux reflètent ce mécanisme. Ici, à partir des adresses 0 et 12, trois mots doubles correspondant à la longueur du télégramme standard 81 sont lus et mémorisés dans DB10 pour retraitement ultérieur.

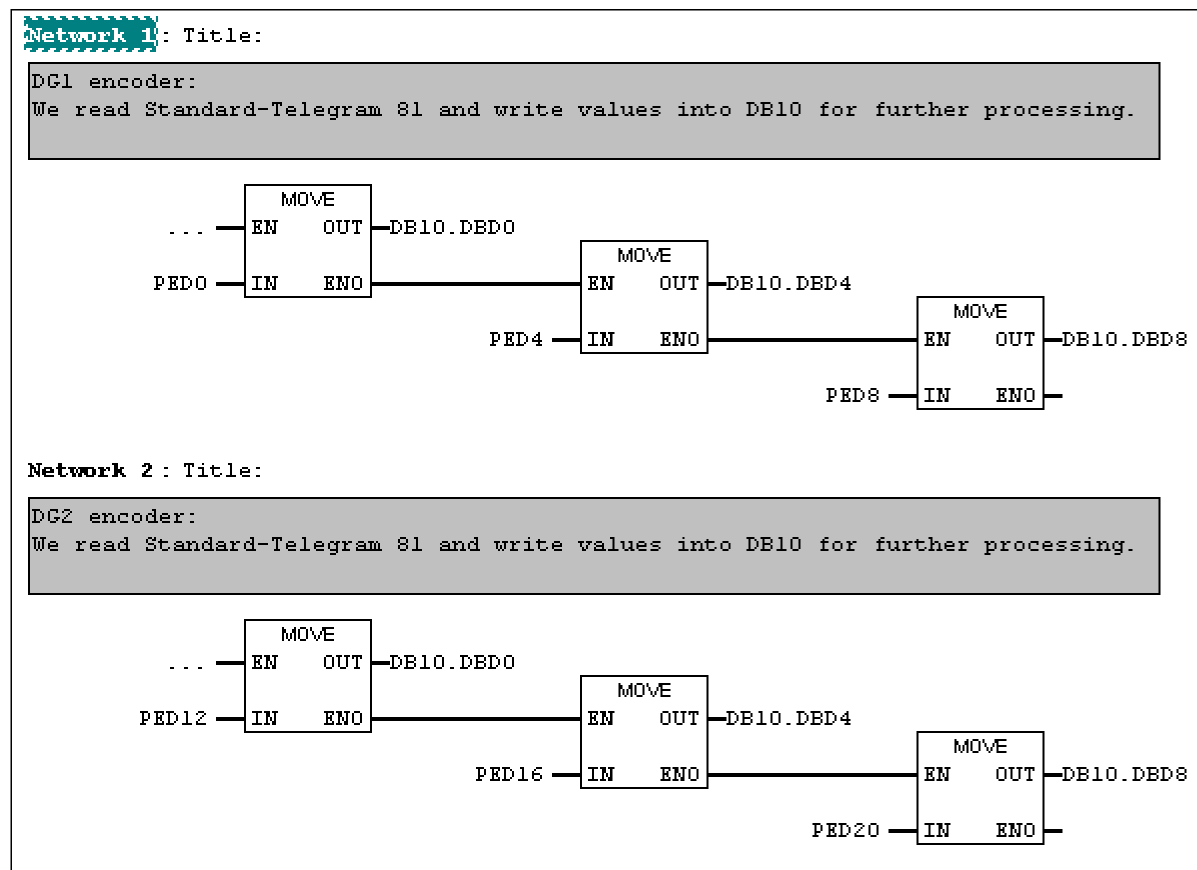


Illustration 9 : Implémentation de OB1. Lecture des valeurs de position.

Déclenchement du prépositionnement

Dans le profil codeur PROFINET, la valeur de prépositionnement est également connue comme le paramètre 65000. Par défaut, la valeur de prépositionnement est nulle, mais elle peut être modifiée à l'aide du dialogue de l'illustration 10. Les conditions préalables à ce changement sont décrites dans le chapitre "Configuration des paramètres utilisateur du codeur".

L'illustration 10 ci-dessous représente le réseau à utiliser pour déclencher un prépositionnement absolu du codeur dg1.

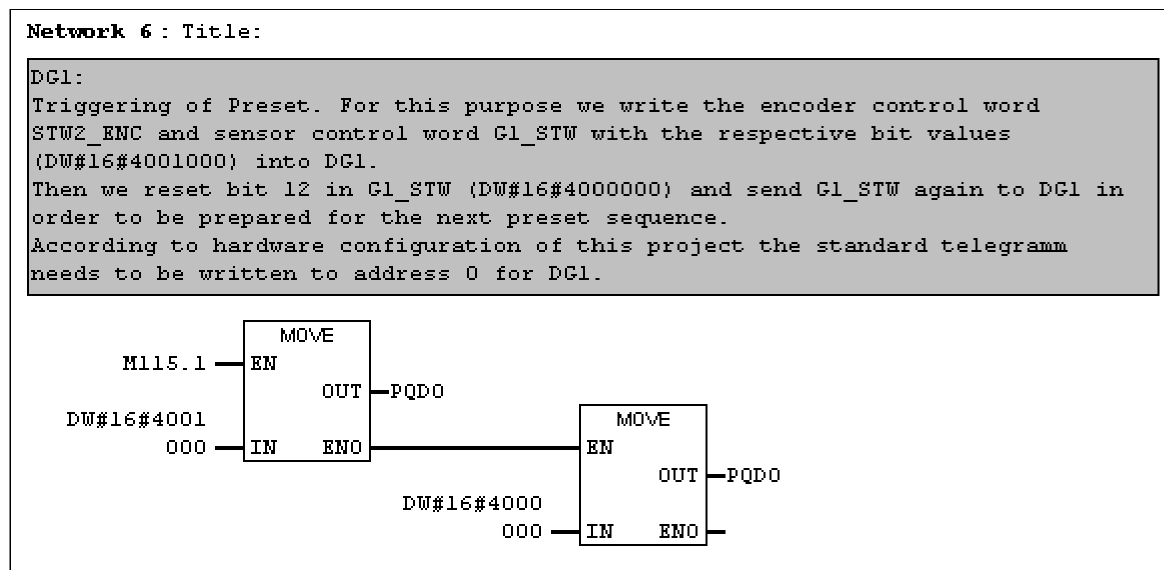


Illustration 10 : Déclenchement du prépositionnement sur dg1

Pour déclencher un prépositionnement absolu sur le codeur, il faut lui envoyer le mot de contrôle codeur STW2_ENC avec le bit API en position 10 mis à 1 et le mot de contrôle capteur G1_STW avec le bit en position 12 mis à 1.

Pour pouvoir effectuer un nouveau prépositionnement, il faut envoyer au codeur G1_STW avec le bit en position 12 remis à zéro.

L'illustration 10 s'applique au codeur 2.

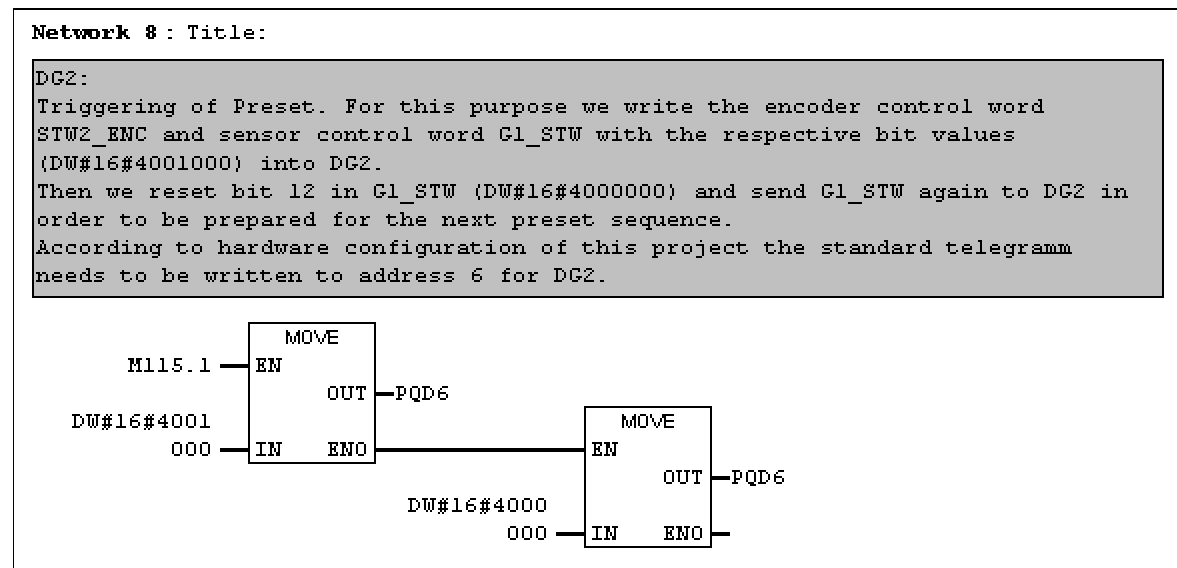


Illustration 11 : Déclenchement du prépositionnement sur dg2

Dans cet exemple, le prépositionnement lui-même est réalisé en mettant à 1 le bit 1 de M115. Ceci est représenté sur l'illustration 12 sous la forme d'un tableau de variables. Dans le cas d'une connexion en ligne entre le PC et l'API, ce bit peut être mis à 1 et à 0.

Pour plus de détails sur le codeur, voir référence [2].

Dans ce cas particulier, le tableau des variables est utilisé uniquement afin de montrer son fonctionnement. Sur le terrain, il est possible qu'il y ait des commutateurs réels (hardware) qui commandent le déclenchement du prépositionnement.

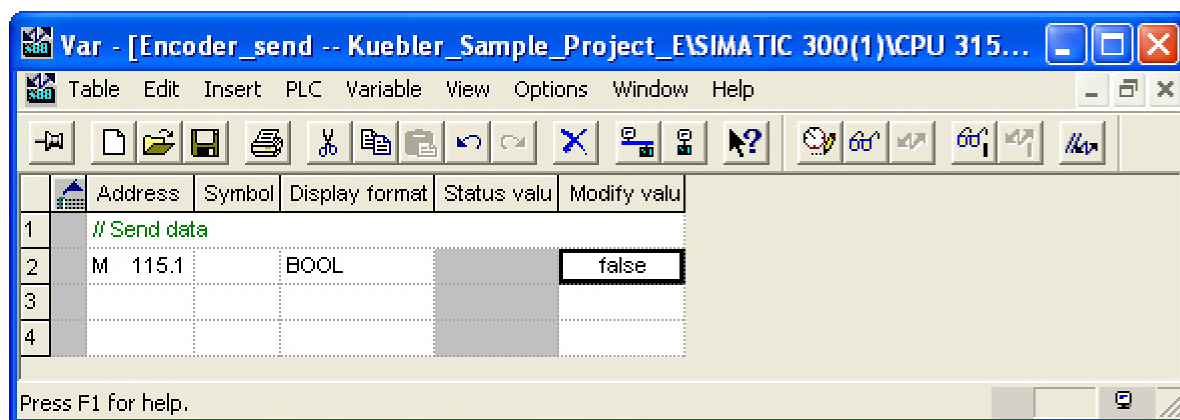


Illustration 12

Téléchargement du paramètre 65000 Prépositionnement

Le chapitre précédent a décrit le déclenchement du prépositionnement. Ce chapitre décrit le téléchargement de la valeur du paramètre de prépositionnement elle-même.

Lors du téléchargement, la valeur de prépositionnement n'est acceptée que si elle répond aux critères suivants :

0 ≤ Prépositionnement ≤ TMR (SubIdx 10 Paramètre 65001) et Prépositionnement ≤ MaxINT32

Conformément à la spécification, la valeur de prépositionnement se base sur des unités affectées d'une échelle ; il faut donc mettre à 1 le bit de contrôle d'état de fonctionnement "Scaling function control" (contrôle de la fonction d'échelle), ainsi que "Class 4 functionality" (fonctionnalité de classe 4). De même, il faut régler correctement les deux valeurs Measuring Units per revolution (MUR – unités de mesure par tour) et Total Measuring Range (TMR – plage de mesure totale) à l'avance.

La figure 8 montre comment envoyer la valeur de prépositionnement au codeur lors du démarrage du système.
Il existe trois autres manières d'envoyer la valeur de prépositionnement au codeur. Elles sont décrites dans les chapitres suivants.

Téléchargement de la valeur de prépositionnement à l'aide du bloc Kuebler FB1 STEP7
Une bibliothèque spécifique à Kübler appelée "Kuebler_Library.zip" est disponible sur notre serveur Intenet. Le bloc fonctionnel FB1 permet d'envoyer à tout moment la valeur de prépositionnement au codeur depuis le programme de l'API.
La bibliothèque Kübler s'installe comme décrit dans les illustrations 13 à 15. Il faut en particulier enregistrer et décompacter la bibliothèque dans le dossier S7libs du dossier d'installation de STEP7.

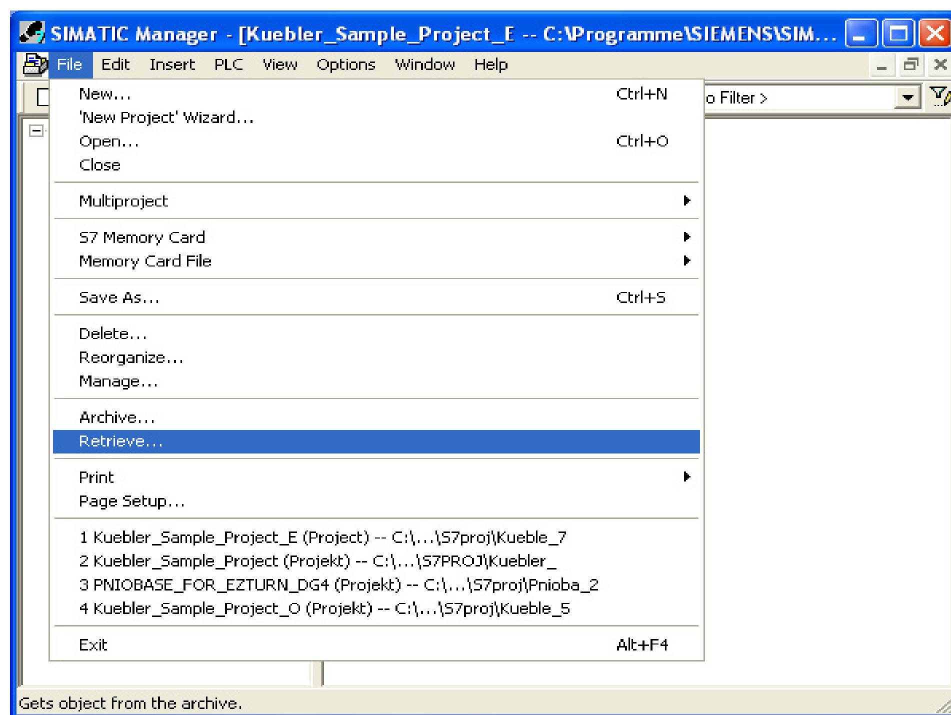


Illustration 13

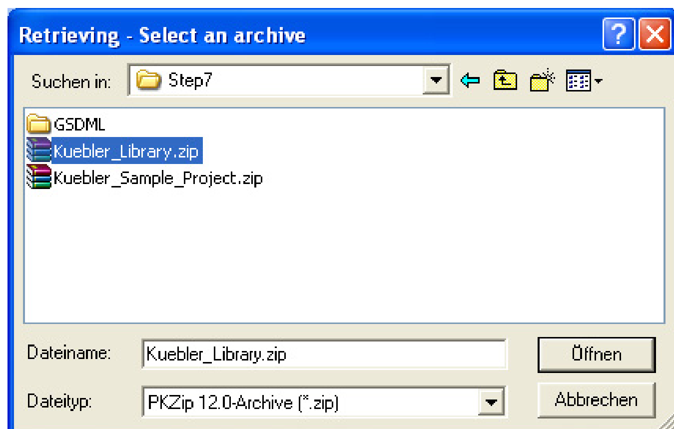


Illustration 14



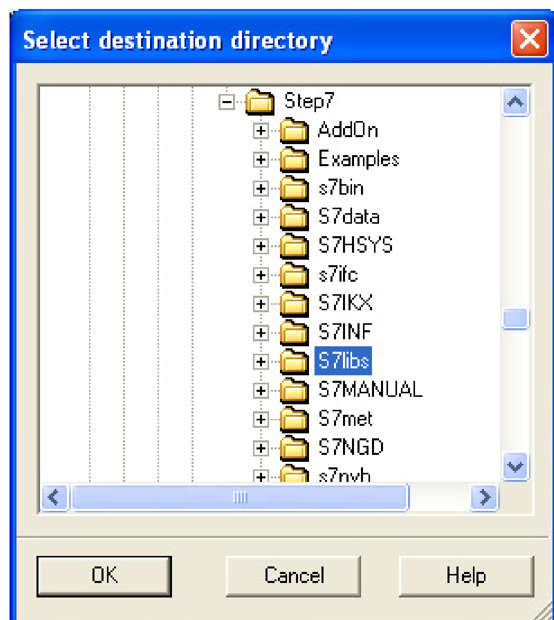


Illustration 15

La bibliothèque et ses composants deviennent visibles dans l'éditeur de blocs, dans l'onglet des éléments de programme lorsqu'elle a été installée avec succès. Ceci est représenté dans l'illustration 16. Il est dorénavant possible d'utiliser le bloc fonctionnel "FB1 PRESET_ENCODER Kuebler" en le glissant et en le déposant dans n'importe que autre bloc créé par le programmeur.

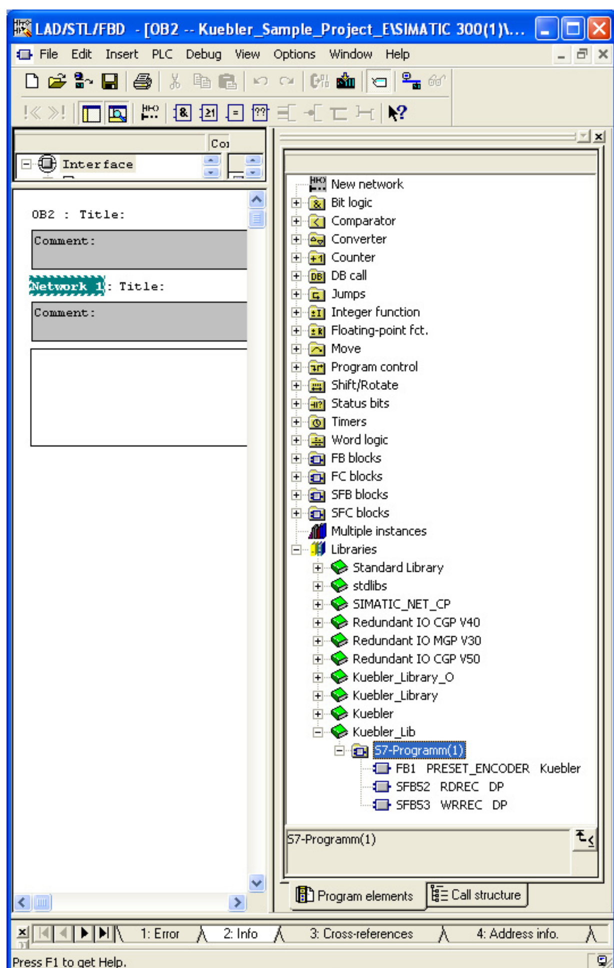


Illustration 16

Toujours sur la base du projet "Kuebler_Sample_Project", les illustrations 17 à 18 montrent l'utilisation des blocs fonctionnels de la bibliothèque Kuebler. Les commentaires ajoutés à l'implémentation du réseau même dans les quatre figures sont importants. Ils décrivent en particulier les paramètres d'entrée et de sortie des blocs.

Il est important de se souvenir que le bloc fonctionnel de la bibliothèque Kuebler se contente de télécharger la valeur de prépositionnement, et ne déclenche pas la fonction de prépositionnement même.

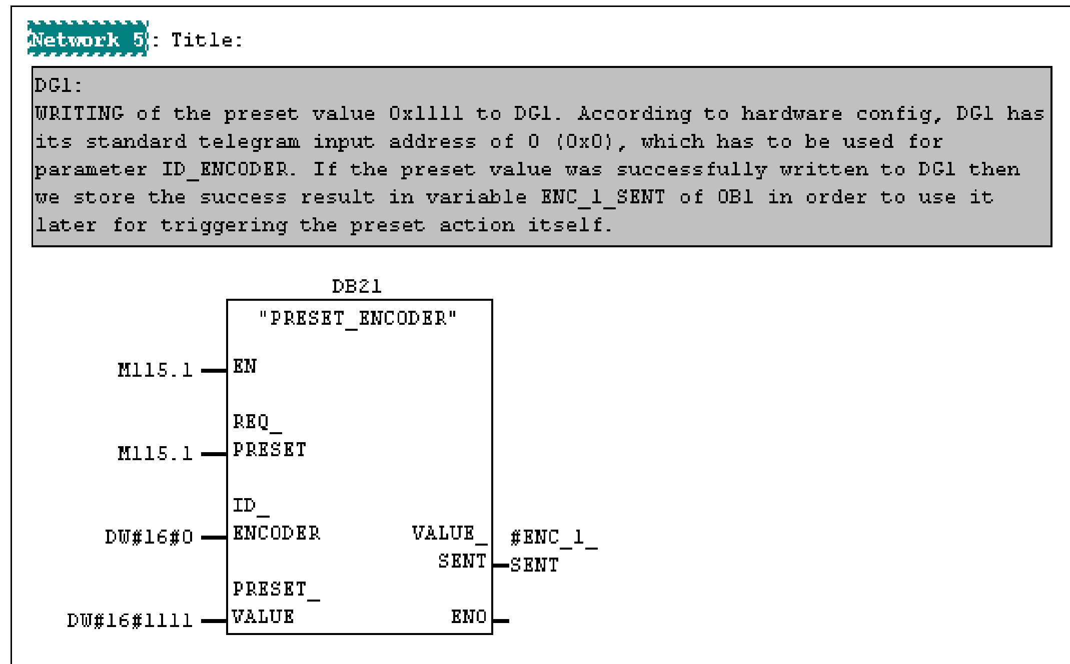


Illustration 17

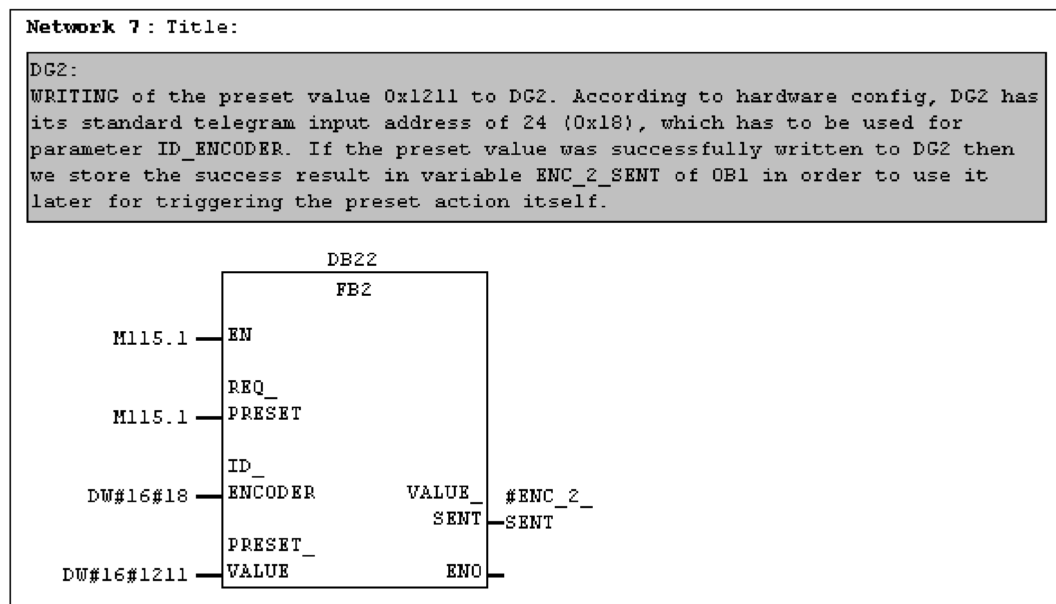


Illustration 18

Ecriture de la valeur de prépositionnement à l'aide de l'application Ezturn
Le logiciel Ezturn, avec sa documentation, peut être téléchargé depuis notre serveur Internet. Le programme d'installation se trouve dans le dossier "Ezturn CANopen_ProfiNet_RS485". Double-cliquer sur CDStart.exe pour lancer l'installation.

L'écriture de la valeur de prépositionnement sur le codeur à l'aide de Ezturn est très simple; elle consiste en deux étapes.

1. Saisir la valeur du prépositionnement dans le champ d'édition "Preset control" de l'onglet Monitor. Cette valeur est de 5789 dans l'exemple ci-dessous.
2. Presser le bouton "Write preset value to encoder" (écrire la valeur de prépositionnement sur le codeur) ; cette action mémorise la valeur de prépositionnement de manière persistante (non effacée par un reset) ET déclenche l'action de prépositionnement même.

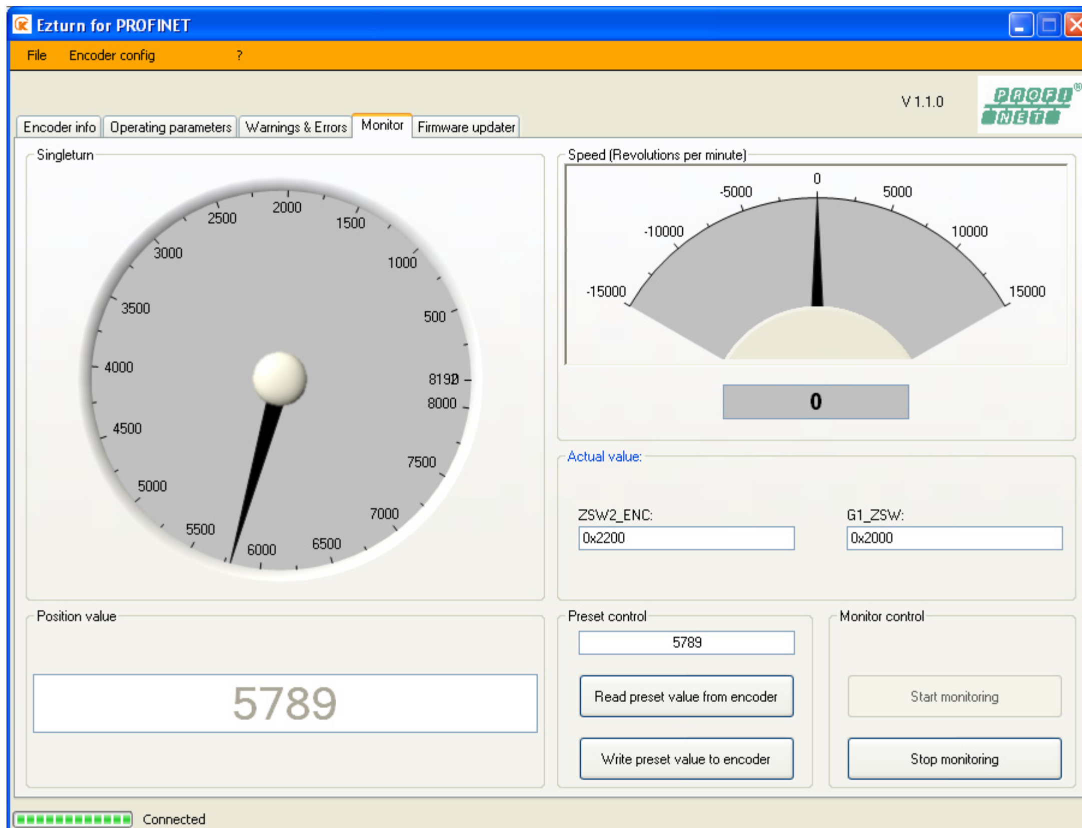


Illustration 19 : Ecriture et activation du prépositionnement à l'aide de l'application Ezturn

Ecriture de la valeur de prépositionnement par programmation en langage C
Cette méthode s'applique à tous les maîtres programmés en langage C. Elle s'applique particulièrement aux maîtres à base de contrôleurs CP1616 installés dans un PC et fonctionnant sous Linux – RTAI.

L'Annexe A présente cette méthode en montrant les diagrammes de séquence UML respectifs, ainsi que le code source lui-même.

L'application codeur

L'application codeur est conforme aux spécifications suivantes :

- Profile Encoder, Technical Specification for Profibus and PROFINET related to PROFIdrive Version 4.1 December 2008 (Profil Codeur, Spécification technique pour Profibus et PROFINET associée à PROFI drive version 4.1 Décembre 2008).
  N° de commande : 3.162
- Profile Drive Technology PROFIdrive Technical Specification for PROFIBUS and PROFINET Version 4.1 May 2006 (Profil Technologie des entraînements PROFIdrive Spécification technique pour PROFIBUS et PROFINET Version 4.1 Mai 2006.
  N° de commande : 3.172.

MRP PROFINET

A partir du firmware version 2.00, le codeur comporte également la fonctionnalité MRP (Media Redundancy Protocol) Pour davantage d'informations sur les caractéristiques de MRP, reportez-vous à la littérature correspondante et à l'Internet. Fondamentalement, l'avantage de MRP réside dans le fait que la fonctionnalité des composants, qui sont câblés selon une structure en anneau comme représenté ci-dessous, est maintenue en cas de défaillance ou de rupture des câbles à quelque endroit que ce soit.

Dans l'exemple concret de l'illustration 20 ci-dessous, la structure en anneau est reconfigurée par la commande en une topologie linéaire en cas de rupture du segment A ou C. L'échange d'informations avec les deux codeurs a alors lieu respectivement via l'autre port de la commande. Les chiffres 1 et 2 représentent les numéros de port respectifs de l'appareil concerné.
En cas de rupture du segment B, la topologie en anneau est reconfigurée en deux topologies linéaires, chacune commandant un codeur.

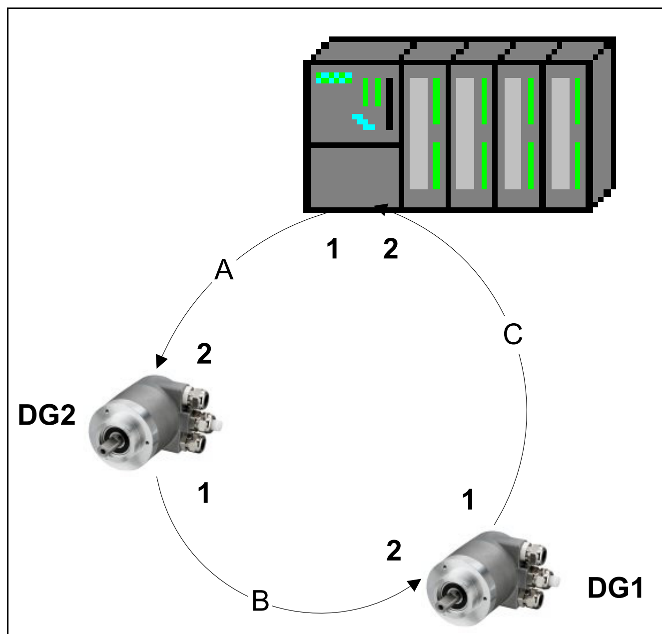


Illustration 20

Configuration d'un projet MRP

Le chapitre "Exemple de configuration d'un projet à l'aide de STEP7" montre le fonctionnement des deux codeurs sur la CPU315-2PN/DP.

Les pages suivantes représentent la configuration des trois composants pour le fonctionnement MRP.



Important !



La fonctionnement du codeur MRP exige impérativement l'installation du nouveau fichier GSDML GSDML-V2.2-KUEBLER-0198-Sendix 58xxPNIO-20130116-131800.xml. Sur l'illustration représentant le catalogue du hardware, l'entrée indiquée par la flèche rouge représente le codeur MRP, et donc ce fichier GSDML.

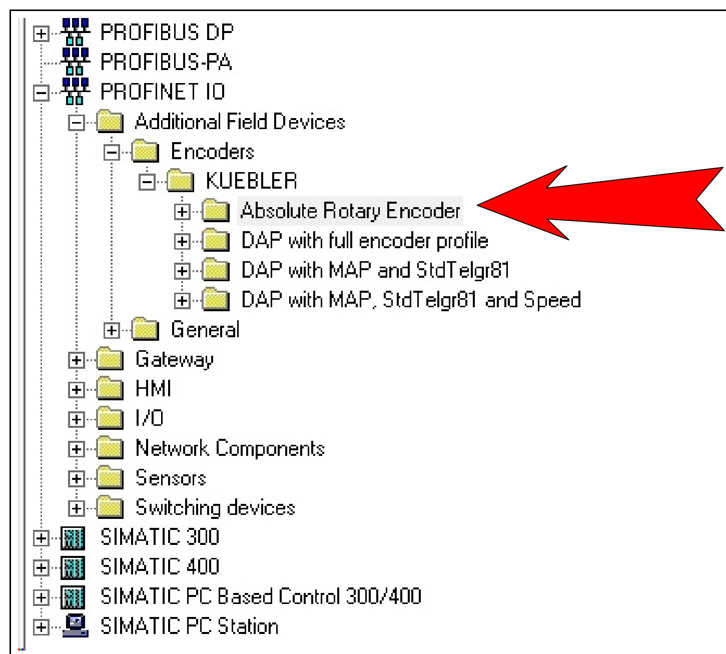


Illustration 21

Les trois autres entrées au-dessous de celle indiquée par la flèche représentent les DAP des codeurs non-MRP. Un double clic sur l'icône du codeur dans la configuration hardware sous STEP7 ouvre la fenêtre de dialogue représentée dans l'illustration 22 ci-dessous, qui montre la version de firmware du codeur et la nom du fichier GSDML. Pour le codeur MRP, la version de firmware doit être au moins 200.



Properties - dg1

General | Identification

Short description: sendix58xx

Sendix 58xx PNIO

Order No./ firmware: 8.58x8.xxC2.C112 / V200

Family: KUEBLER

Device name: dg1

GSD file: GSDML-V2.2-KUEBLER-0198-Sendix58xxPNIO-20130116-131800.xml

Change Release Number...

Node in PROFINET IO System

Device number: 1 PROFINET-IO-System (100)

IP address: 192.168.0.12 Ethernet...

☑ Assign IP address via IO controller

Comment:

OK Cancel Help

Illustration 22



Configuration de CPU315 2PN/DP pour le fonctionnement MRP

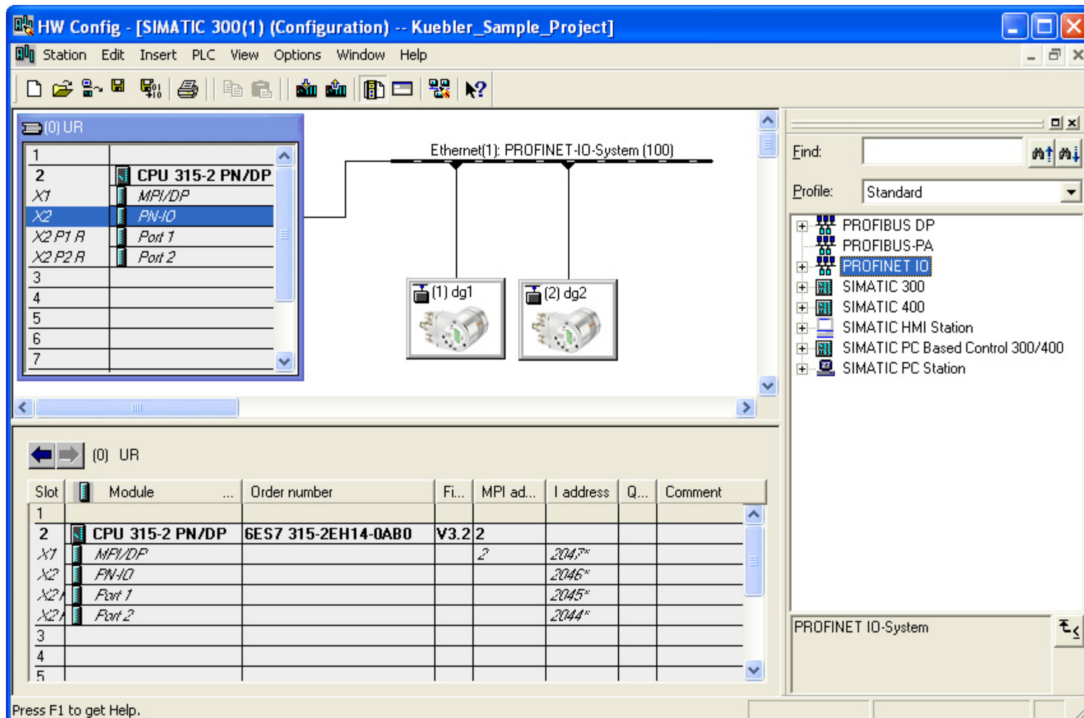


Illustration 23

Comme représenté dans l'illustration 23, double-cliquer sur la ligne “PN-IO”. Dans la fenêtre de dialogue qui s'ouvre, régler tous les paramètres comme indiqué dans l'illustration 24. En particulier, CPU315 doit être configuré comme manager (gestionnaire) MRP.

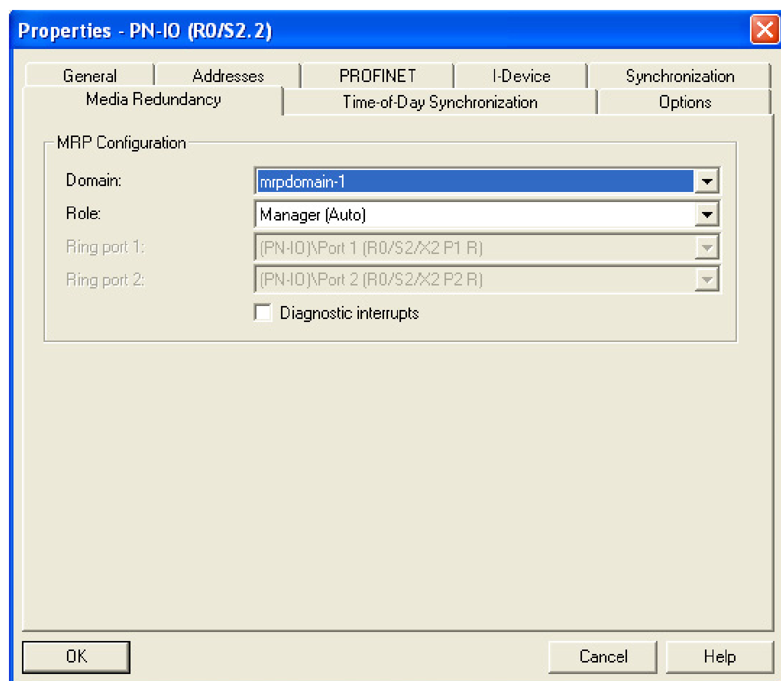


Illustration 24

Double-cliquer sur la ligne "Port1" pour ouvrir la fenêtre de dialogue représentée dans l'illustration 25, dont les paramètres doivent être réglés comme suit. Le port 1 de CPU315 est alors relié au port 2 de DG2.

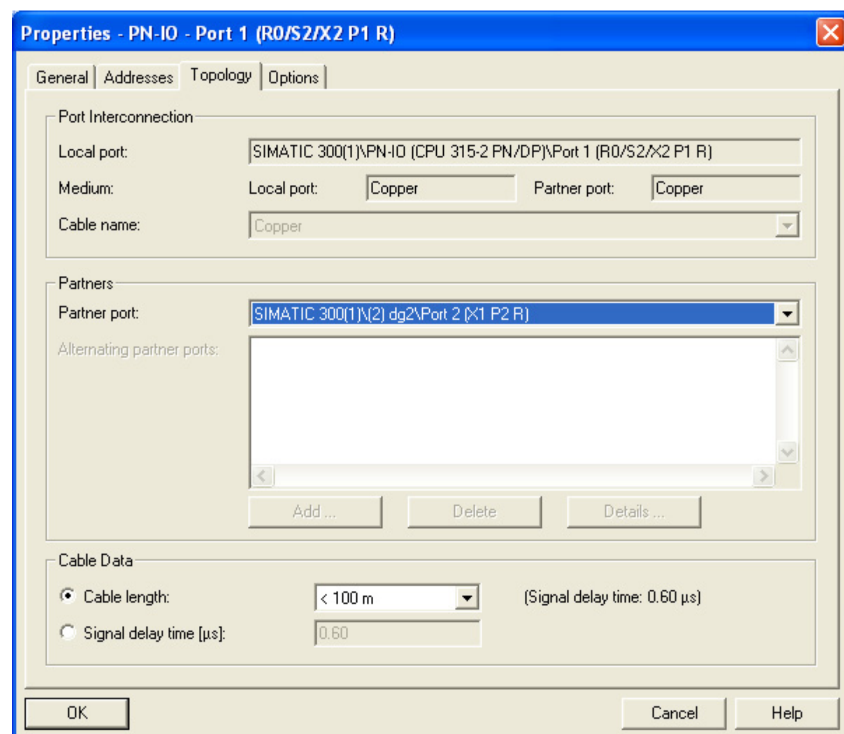


Illustration 25

La fenêtre de dialogue ci-dessous s'applique au port 2. Le port 2 de CPU315 est alors relié au port 1 de DG1.



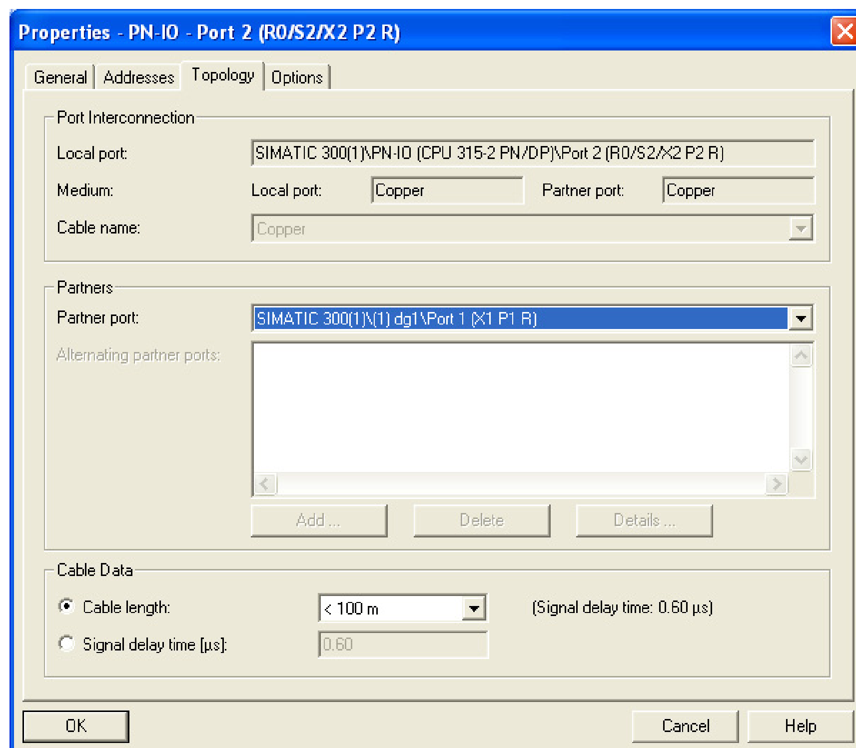


Illustration 26

Configuration des deux codeurs pour le fonctionnement MRP
La configuration précédente s'applique également à DG1, à partir de l'illustration 27.

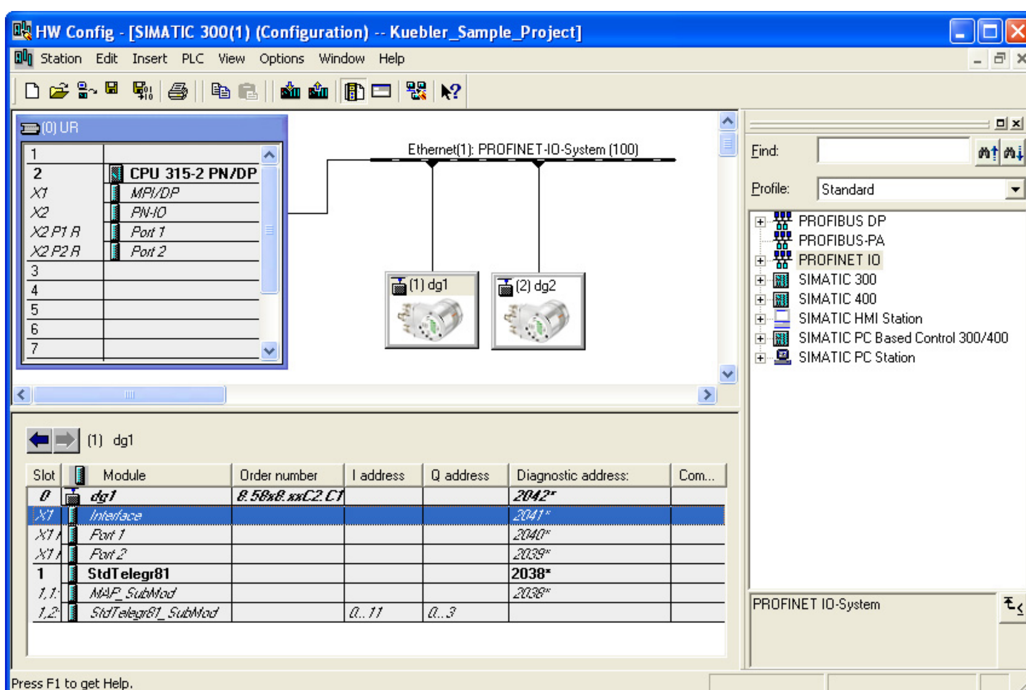


Illustration 27

Double-cliquer sur la ligne "Interface" de DG1 pour ouvrir la fenêtre de dialogue représentée dans l'illustration 28. DG1 est un client MRP et doit être paramétré en tant que tel.



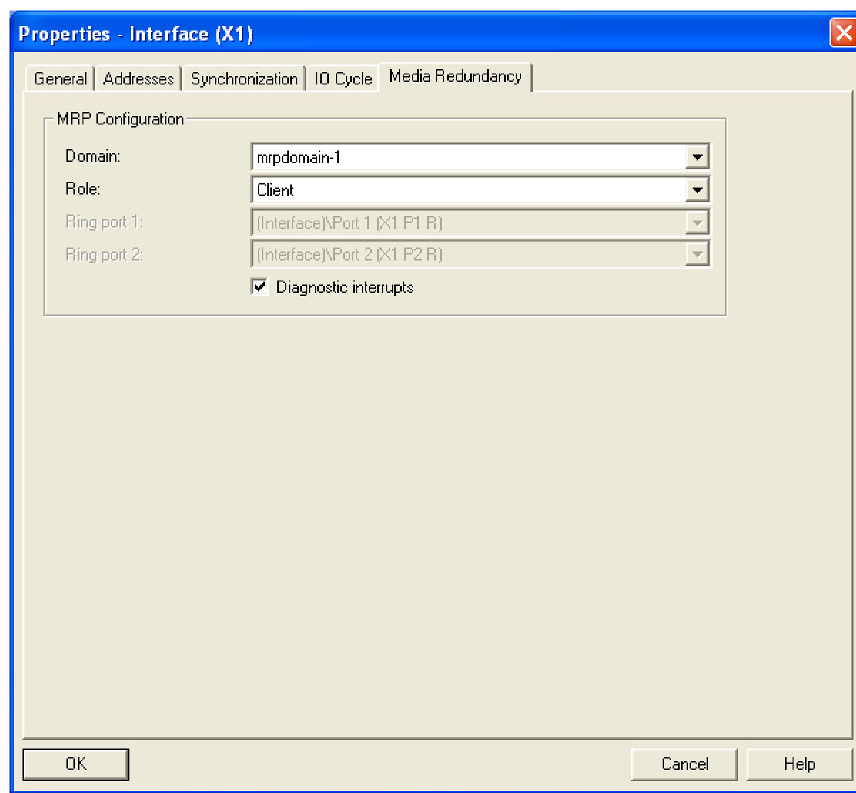


Illustration 28

Les réglages des illustrations 29 et 30 s'appliquent aux deux ports 1 et 2 de DG1.

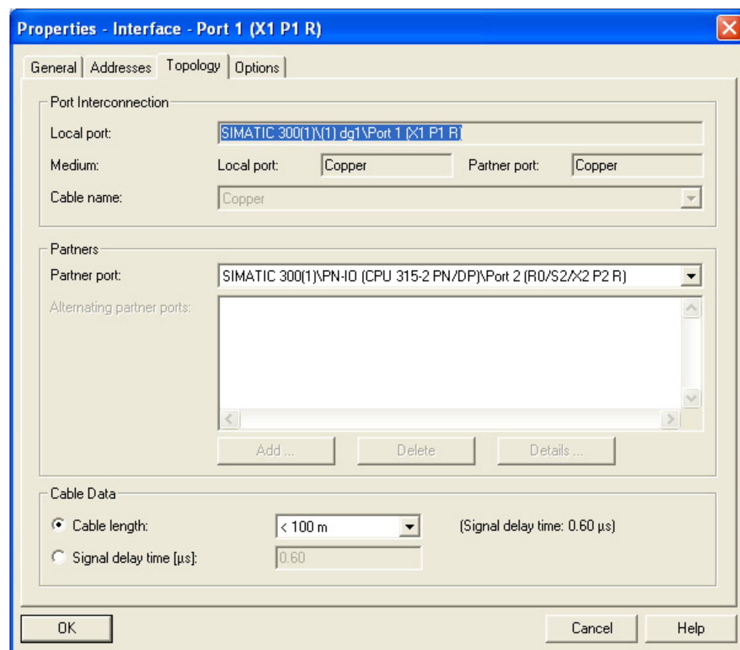


Illustration 29

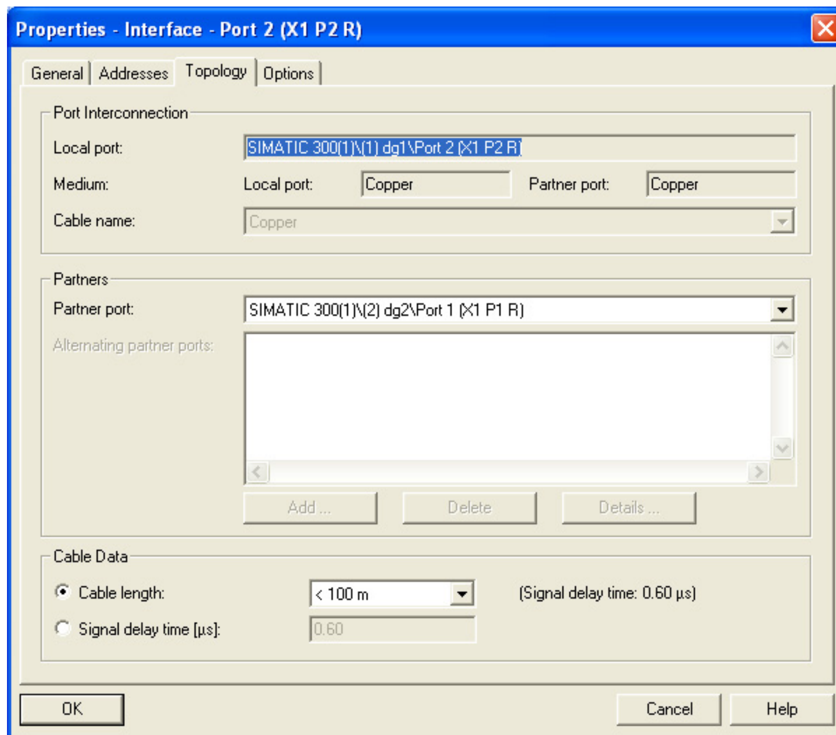


Illustration 30

Comme DG1, DG2 doit être configuré comme client MRP.

Après un téléchargement via le port 1 de CPU315, le port 1 est relié au port 2 de DG2. Il faut alors redémarrer CPU315 au moyen du bouton reset.

En cas d'interruption de l'anneau, p. ex. au niveau du port 1 de la CPU, cette dernière affiche une erreur au moyen de la LED correspondante, mais elle ne passe pas en mode Stop. Le message d'erreur disparaît dès que l'anneau est refermé.

En cas d'interruption de l'anneau, il est possible de brancher le PC STEP7 et de réaliser un diagnostic d'erreur comme suit.

Comme représenté dans l'illustration 31, sélectionner la ligne "PN-IO" de CPU315 et, dans le menu "PLC", sélectionner l'option “Module Information…”.



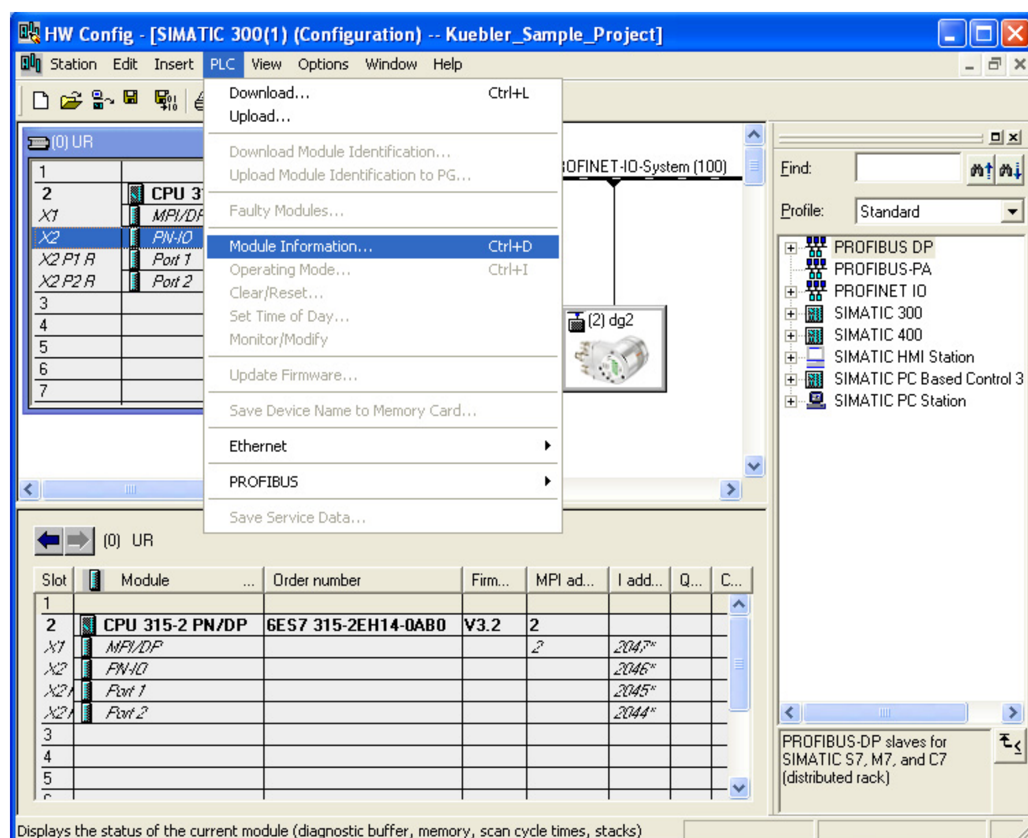


Illustration 31

Une fenêtre de dialogue s'ouvre alors, représentant à la fois l'état du composant et le staut de la communication. Ces informations sont représentées dans les illustrations 32 et 33.

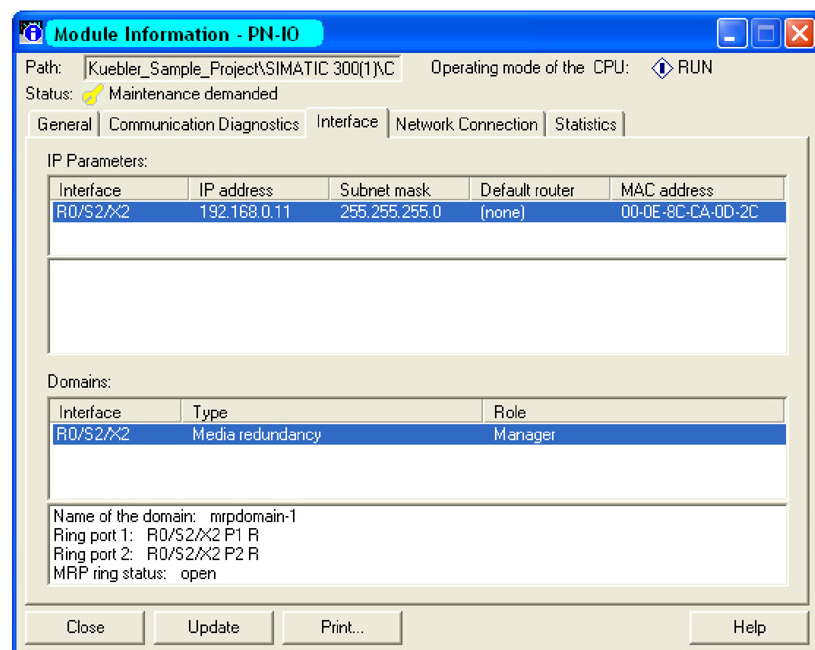


Illustration 32



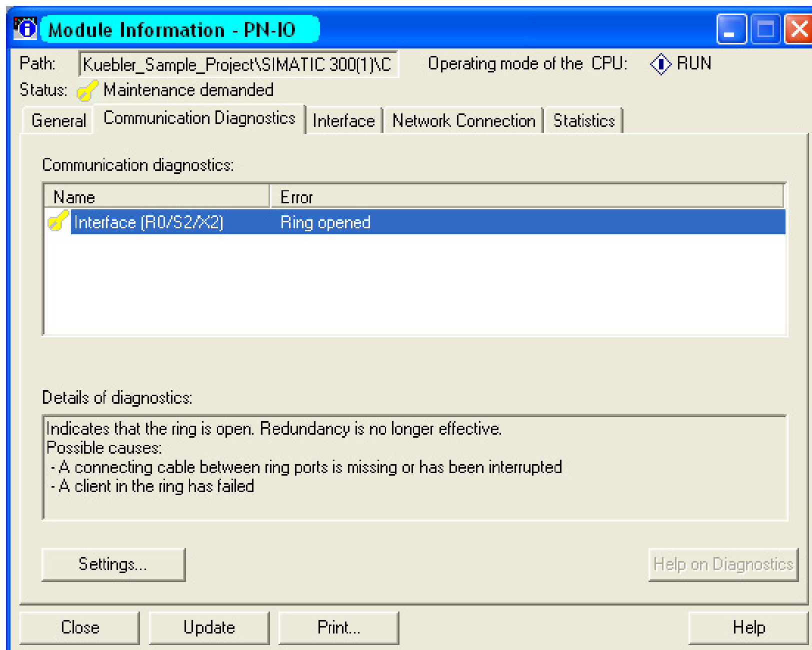


Illustration 33

En alternative, une vue générale de la fenêtre de dialogue peut être obtenue en sélectionnant le menu “PROFINET IO Topology...”. Cette fenêtre, représentée dans l'illustration 35, montre que le port 1 de la CPU et le port 2 de DG2 présentent une erreur.

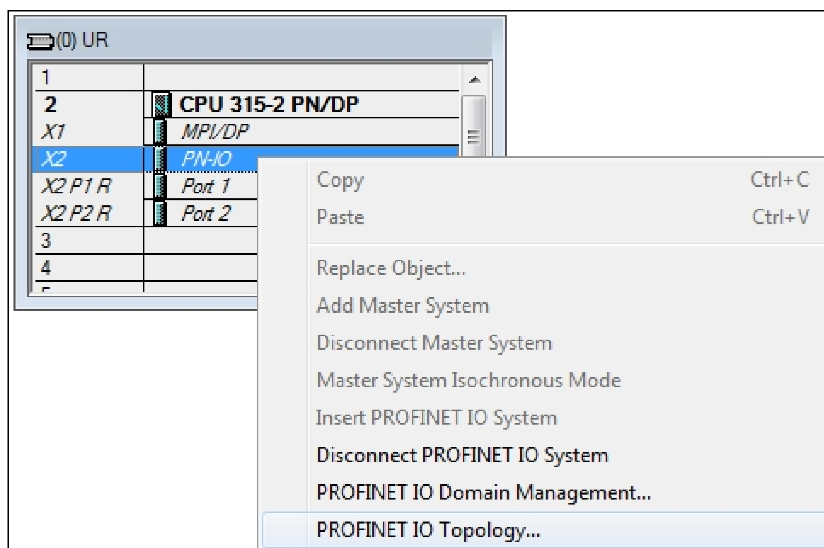


Illustration 34

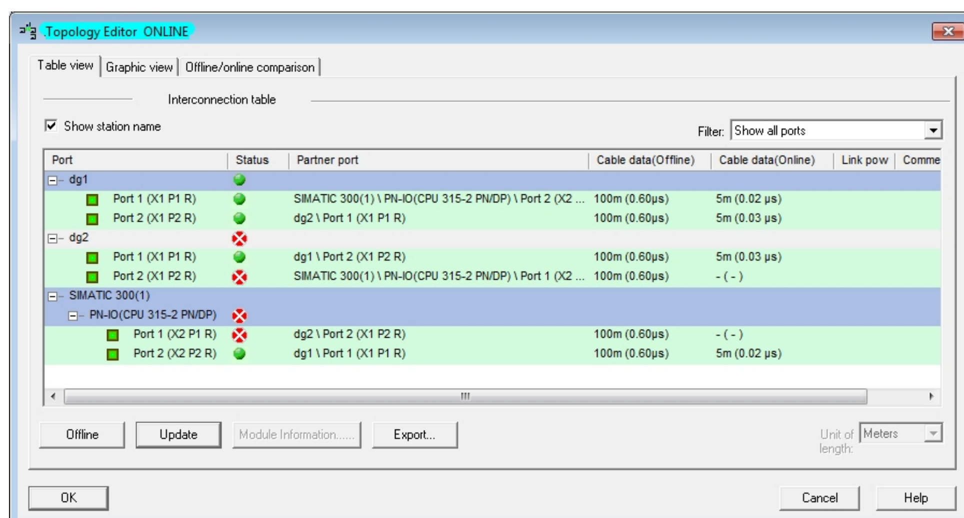


Illustration 35

Annexe A : Lecture/écriture de la valeur de présélection Prm 65000

Le mécanisme d'écriture / de lecture du paramètre de prépositionnement 65000 est connu sous le nom de "Base Mode Parameter Access" (BMPA – Mode d'accès de base aux paramètres) ; il est décrit en détail aux pages 59 et suivantes de la spécification du profil PROFIdrive.

L'illustration 36 décrit un diagramme de séquence de message pour la lecture du paramètre 65000. Toutes les fonctions dont le nom commence par "kblr_" sont spécifiques à Kuebler ; ces noms peuvent être remplacés par des noms de fonction spécifiques à l'utilisateur. Tous les autres noms de fonctions correspondent à des fonctionnalités de PROFINET – API pour le contrôleur CP1616 définies par SIEMENS et ne peuvent pas être modifiés.

La requête de lecture du paramètre 65000 dans le cadre de l'accès BMPA est en fait une requête d'écriture suivie d'une requête de lecture.

La fonction kblr_readPrm_65000_Preset remplit tous les structs selon BMPA et les transmet ensuite à la fonction PNIO_rec_write_req.

Les extraits de code correspondants sont donnés dans les pages suivantes. Une situation similaire apparaît dans le cas d'une demande d'écriture du paramètre 65000 dans le cadre de l'accès BMPA, qui est une combinaison d'une requête d'écriture suivie d'une requête de lecture à l'index BMPA.



Illustration 36

Les structs sont les suivants :

```
typedef struct
{
	PNIO_UINT8	RequestID;
	PNIO_UINT8	RequestRef;
	PNIO_UINT8	NoOfPrms;
	PNIO_UINT8	DO_ID;

} sBMPReqHeader;

typedef struct
{
	PNIO_UINT8	NoOfElements;
	PNIO_UINT8	Attribute;
	PNIO_UINT16	PNU;
	PNIO_UINT16	SubIdx;
} sBMPPrmAddress;

typedef struct
{
	PNIO_UINT8	NoOfValues;
	PNIO_UINT8	Format;
} sBMPPrmValue;

typedef struct
{
	PNIO_UINT8	ResponseID;
	PNIO_UINT8	RequestRefMir;
	PNIO_UINT8	NoOfPrms;
	PNIO_UINT8	DO_IDMir;
}sBMPRespHeader;

typedef struct
{
	sBMPRespHeader	BMPRespHeader;
	sBMPPrmValue	BMPPrmValue;
	PNIO_UINT8	valArray [sizeof (PNIO_UINT32)];
} sBMPResponseBuf;

typedef enum
{
	KBLR_REQUEST_PARAMETER	=	1,
	KBLR_CHANGE_PARAMETER	=	2
} KBLR_RequestID;

typedef enum
{
	KBLR_VALUE	=	0x10,
	KBLR_DESCRIPTION	=	0x20,
	KBLR_TEXT	=	0x30
} KBLR_Attribute;
```

```
typedef enum
{
	KBLR_ZERO	=	0x40,
	KBLR_BYTE	=	0x41,
	KBLR_WORD	=	0x42,
	KBLR_DWORD	=	0x43,
	KBLR_ERROR	=	0x44
} KBLR_Format;

typedef enum
{
	KBLR_REQUEST_PARAMETER_P	=	0x01,
	KBLR_CHANGE_PARAMETER_P	=	0x02,
	KBLR_REQUEST_PARAMETER_M	=	0x81,
	KBLR_CHANGE_PARAMETER_M	=	0x82
} KBLR_ResponseID;

#define KBLR_BASEMODEPRMACCESS_INDEX 0xB02E

/****************************************************************** */
/* Cette fonction lit de manière acyclique le paramètre			*/
/* Prm_65000_Preset depuis le codeur						*/
/****************************************************************** */

void kblr_readPrm_65000_Preset (void)
{
	sBMPReqHeader		BMPReqHeader;
	sBMPPrmAddress		BMPPrmAddress;
	PNIO_UINT8*		pMem8 = NULL;

	// L'adresse logique du sous-module MAP/PAP est 0
	// dans cet exemple
	PNIO_ADDR		SubModAddress = { PNIO_ADDR_LOG, PNIO_IO_OUT, 0 };
	PNIO_UINT32		dwErrorCode;
	PNIO_UINT32		RecordIndex = KBLR_BASEMODEPRMACCESS_INDEX;
	PNIO_REF		ReqRef = 1;

	// Requête BMPA pour un struct à valeur unique
	BMPReqHeader.RequestID	=	KBLR_REQUEST_PARAMETER;
	BMPReqHeader.RequestRef	=	0xAB; // Doit être réfléchi par le codeur
	BMPReqHeader.NoOfPrms	=	0x01
	BMPReqHeader.DO_ID	=	0xCD; // Doit être réfléchi par le codeur

	BMPPrmAddress.NoOfElements	=	0x00;
	BMPPrmAddress.Attribute	=	KBLR_VALUE;
	BMPPrmAddress.PNU	=	OsHtons(65000); // BIG ENDIAN !

	pMem8 = (PNIO_UINT8*)malloc(sizeof (sBMPReqHeader)+
					sizeof (sBMPPrmAddress));

	memcpy(pMem8,(PNIO_UINT8*)&BMPReqHeader, sizeof (sBMPReqHeader));
	memcpy(pMem8 + sizeof (sBMPReqHeader),(PNIO_UINT8*)&BMPPrmAddress,
					sizeof (sBMPPrmAddress));
```

```
dwErrorCode = PNIO_rec_write_req (
                    g_dwHandle,       // handle
                    &SubModAddress, // Adresse du sous-module
                    RecordIndex,
                    ReqRef,
                    sizeof (sBMPReqHeader) + sizeof (sBMPPrmAddress),
                    (PNIO_UINT8*)pMem8);
    free(pMem8);
}
```

Lors de l'exécution de callback_for_ds_write_conf, il faut examiner le code d'erreur retourné afin de décider si la valeur requise peut déjà être "récupérée" au moyen d'une requête de lecture BMPA.

La fonction ci-dessous décrit cette procédure :

```
/*********************************************************************/
/* Cette fonction doit être appelée immédiatement après
kblr_readPrm_65000_Preset afin de lire le résultat de
la requête 'parameter request' pour le paramètre 65000_Preset. Il
/* suffit de lire dans l'index 0xB02E qui est l'index BMPA
/*********************************************************************/

void kblr_readResponse_readPrm65000Preset (void)
{
    // L'adresse logique du sous-module MAP/PAP
    // est 0 dans cet exemple.
    PNIO_ADDR         SubModAddress = { PNIO_ADDR_LOG, PNIO_IO_OUT, 0 };
    PNIO_UINT32       dwErrorCode;
    PNIO_UINT32       RecordIndex = KBLR_BASEMODEPRMACCESS_INDEX;
    PNIO_REF          ReqRef = 1;
    dwErrorCode = PNIO_rec_read_req(
                    g_dwHandle,          // handle
                    &SubModAddress,      // Adresse du sous-module
                    RecordIndex,
                    ReqRef,
                    sizeof (sBMPResponseBuf));
}
```

Comme callback_for_ds_read_conf fournit un pointeur pour les données des paramètres et le code d'erreur, le cycle de requête et de réponse s'achève. Le texte ci-dessous représente un extrait du code de callback_for_ds_read_conf.

```
Case KBLR_BASEMODEPRMACCESS_INDEX:
{
	printf („\r\ncallback_for_ds_read_conf: Réception données d'accès BMP :
		0x%04x\n“, pCbfPrm->RecWriteConf.RecordIndex);

	if (pCbfPrm->RecReadConf.Err.ErrCode == 0xDE)
	{
		printf („\r\nPas de réponse mode BMPA disponible pour l'instant !\r\n“);
	}
	if ( (pCbfPrm->RecReadConf.Err.ErrCode == 0) &&
		(pCbfPrm->RecReadConf.Length > 0))
	{
		for (i=0; i<pCbfPrm->RecReadConf.Length; i++)
	{
		// Simple pointage vers les octets reçus
		printf („pBuf[%02d]=%02x\t“,i,
					*(pCbfPrm->RecReadConf.pBuffer+i));
		}
		printf(„\r\n“);
	}
}
break;
```

Important !



La fonction callback_for_ds_read_conf doit revenir le plus rapidement possible. Il faut impérativement éviter les opérations nécessitant du temps comme printf et les déplacer dans des threads s'exécutant dans un contexte de priorité plus basse.

La figure 37 représente un diagramme de séquence décrivant le scénario de l'écriture du paramètre 65000.

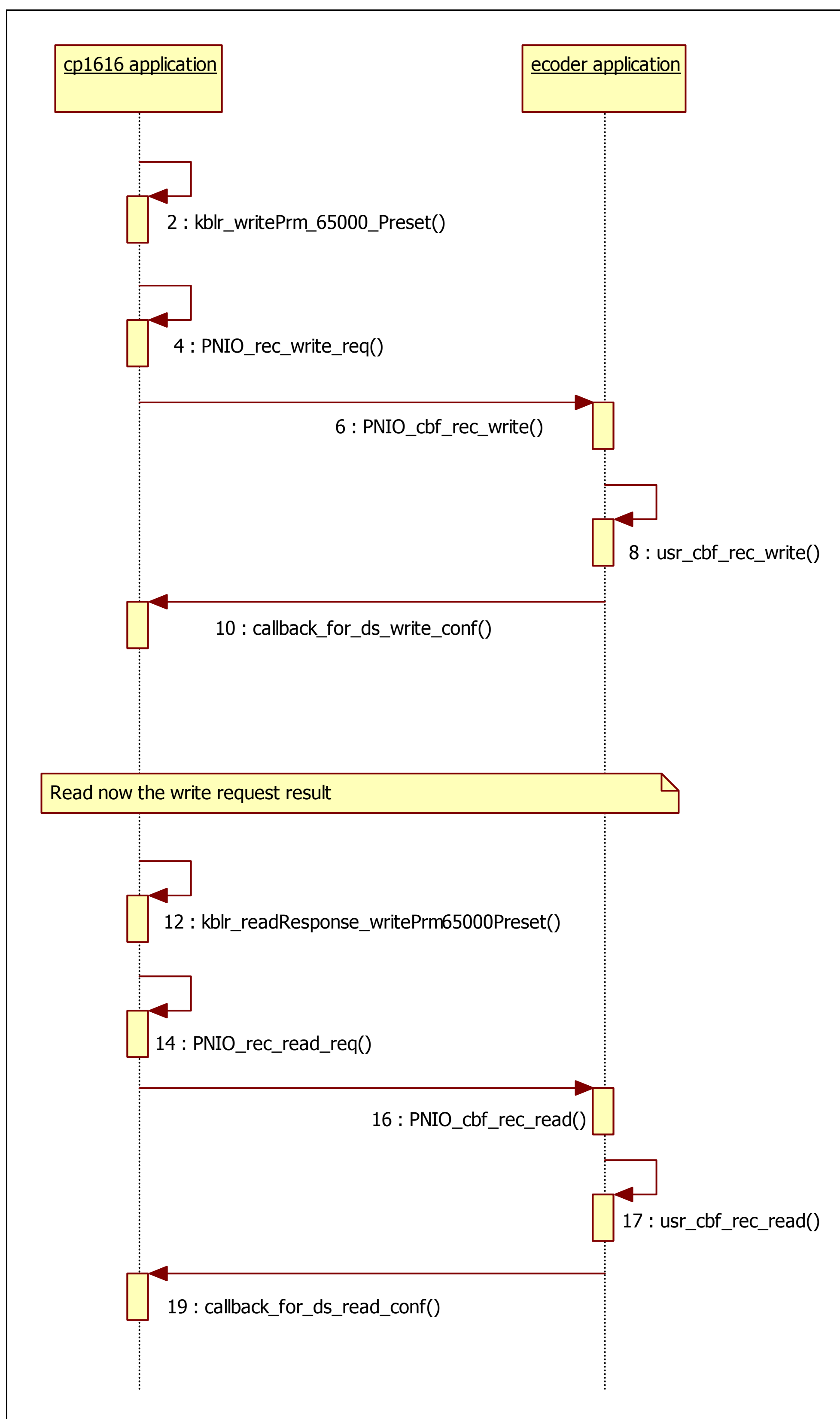


Illustration 37

```
/*********************************************************************/
/* Cette fonction écrit de manière acyclique le paramètre             */
   Prm_65000_Preset dans le codeur                                     */
/*********************************************************************/

void kblr_writePrm_65000_Preset (int32_t i32Preset)
{
	sBMPReqHeader		BMPReqHeader;
	sBMPPrmAddress		BMPPrmAddress;
	sBMPPrmValue		BMPPrmValue;
	PNIO_UINT8*		pMem8 = 0;

	// L'adresse logique du sous-module MAP/PAP
	// est 0 dans cet exemple.
	PNIO_ADDR		SubModAddress = { PNIO_ADDR_LOG, PNIO_IO_OUT, 0 };
	PNIO_UINT32		dwErrorCode;
	PNIO_UINT32		RecordIndex = KBLR_BASEMODEPRMACCESS_INDEX;
	PNIO_REF		ReqRef = 1;

	// Requête BMPA pour un struct à valeur unique
	BMPReqHeader.RequestID		= KBLR_CHANGE_PARAMETER;
	BMPReqHeader.RequestRef		= 0x12; // Doit être réfléchi par
					        // le codeur
	BMPReqHeader.NoOfPrms		= 0x01;
	BMPReqHeader.DO_ID		= 0x34;  // Doit être réfléchi par
					        // le codeur
	BMPPrmAddress.NoOfElements	= 0x01;
	BMPReqHeader.DO_ID		= 0x34; // Doit être réfléchi par
					        // le codeur
	BMPPrmAddress.NoOfElements	= 0x00;
	BMPPrmAddress.Attribute		= KBLR_VALUE;
	MPPrmAddress.PNU		= OsHtons(65000); // BIG ENDIAN !

	BMPPrmValue.NoOfValues		= 1;
	BMPPrmValue.Format		= KBLR_DWORD;

	i32Preset = OsHtonl(i32Preset); // BIG ENDIAN !!!

	pMem8 = (PNIO_UINT8*)malloc (sizeof (sBMPReqHeader)+
		sizeof (sBMPPrmAddress) + sizeof (sBMPPrmValue)+
		sizeof (int32_t));

	memcpy(pMem8, (PNIO_UINT8*)&BMPReqHeader,
		sizeof (sBMPReqHeader));
	memcpy(pMem8+sizeof(sBMPReqHeader),
		(PNIO_UINT8*)&BMPPrmAddress, sizeof (sBMPPrmAddress));

	memcpy(pMem8 + sizeof (sBMPReqHeader) + sizeof (sBMPPrmAddress),
		(PNIO_UINT8*)&BMPPrmValue, sizeof (sBMPPrmValue));

	memcpy(pMem8 + sizeof (sBMPReqHeader) + sizeof (sBMPPrmAddress)+
		sizeof (sBMPPrmValue), &i32Preset, sizeof (int32_t));
```

```
	dwErrorCode = PNIO_rec_write_req(
		g_dwHandle,			// handle
		&SubModAddress, // Adresss du sous_module
		RecordIndex,
		ReqRef,
		sizeof (sBMPReqHeader) + sizeof (sBMPPrmAddress) +
		sizeof (sBMPPrmValue) + sizeof (int32_t),
					(PNIO_UINT8*)pMem8);
		free(pMem8);
	}
```

Il est possible dans la fonction callback_for_ds_write_conf d'exploiter un éventuel code d'erreur retourné et de décider de la lecture ou non du résultat de la requête de modification.

```
/********************************************************************/
/* Cette fonction doit être appelée immédiatement après              */
/* kblr_writePrm_65000_Preset afin de lire le résultat de la         */
/* requête 'parameter change request' pour le paramètre 65000_Preset. */
/* Il suffit de lire dans l'index 0xB02E qui est l'index BMPA         */
/********************************************************************/

void kblr_readResponse_writePrm65000Preset (void)
{
	// L'adresse logique du sous-module MAP/PAP
	// est 0 dans cet exemple.
	PNIO_ADDR			SubModAddress = { PNIO_ADDR_LOG, PNIO_IO_OUT, 0 };
	PNIO_UINT32 dwErrorCode;
	PNIO_UINT32 RecordIndex = KBLR_BASEMODEPRMACCESS_INDEX;
	PNIO_REF			ReqRef = 1;
	dwErrorCode = PNIO_rec_read_req(
		g_dwHandle,		// handle
		&SubModAddress,		// Adresse du sous-module
		RecordIndex,
		ReqRef,
		sizeof (sBMPResponseBuf));
}
```

L'extrait du code source de l'indice BMPA de la fonction callback_for_ds_read_conf s'applique également dans ce cas. Cependant, il faut tenir compte du fait que, dans le premier cas, c'est la valeur de prépositionnement elle-même qui est lue, alors que dans le second cas, il s'agit simplement du résultat de la requête de modification.

Références

1. PROFINET Cabling and Interconnection Technology Guideline Version 2.00 March 2007 (Directives PROFINET sur la technologie de câblage et d'interconnexion Version 2.00 mars 2007)
N° de commande : 2.252

2. Profile Encoder, Technical Specification for Profibus and PROFINET related to PROFIdrive Version 4.1 December 2008 (Profil Codeur, Spécification technique pour Profibus et PROFINET associée à PROFI drive version 4.1 Décembre 2008).
N° de commande : 3.162

3. Profile Drive Technology PROFIdrive Technical Specification for PROFIBUS and PROFINET Version 4.1 May 2006 (Profil Technologie des entraînements PROFIdrive Spécification technique pour PROFIBUS et PROFINET Version 4.1 Mai 2006.
N° de commande : 3.172

www.kuebler.com



Kübler Group
Fritz Kübler GmbH.
Schubertstrasse 47
D-78054 Villingen-Schwenningen
Germany
Tél. : +49 7720 3903-0
Télécopie : +49 7720 21564
info@kuebler.com
www.kuebler.com

l'impulsion pour l'automatisme

R.60713.0003_0